Reflection, CSCW, and ConversationBuilder

William J. Tolone, Simon M. Kaplan, and Douglas P. Bogia

Human-Computer Interaction Laboratory
Department of Computer Science
University of Illinois at Urbana-Champaign
1304 W. Springfield Avenue
Urbana, IL 61801, USA
email: {tolone,kaplan,bogia}@cs.uiuc.edu
phone: +1 217 244 0392 fax: +1 217 333 3501

ABSTRACT

This paper reports on our current investigations into *reflection* and *Computer Supported Cooperative Work* (CSCW) systems. A primarily difficulty which must be address by CSCW applications is that collaborative activities are inherently dynamic, often in unexpected ways. This paper, in particular, contains a discussion on how we perceive the use of reflection within our prototype CSCW application, ConversationBuilder (CB), will provide dynamic support to collaborative activities.

KEYWORDS

Reflection, CSCW, Computer Supported Cooperative Work, ConversationBuilder.

INTRODUCTION

This paper reports on our current investigations into *reflection*[13] and *Computer Supported Cooperative Work* (CSCW) systems[5]. CSCW is a multi-disciplinary field which examines ways in which both new and existing computer technologies can be used support individuals and/or groups as they participate in collaborative activities. The range of CSCW applications is very broad as it includes systems which are designed to support (a)synchronous collaborative activities in distributed and/or non-distributed settings. Examples, include shared editors (e.g., DistEdit[10]), shared drawing workspaces (e.g., ClearBoard[6], CaveDraw[12]), information sharing systems (e.g., Lotus Notes[11], Oval[15], gIbis[3]), etc.

A primarily difficulty which must be address by CSCW applications is that collaborative activities are inherently dynamic, often in unexpected ways[1]. Participants join and leave collaborative activities; the goal(s) of these activities change as the activities progress; even the meaning and effect of actions which compose a collaborative activity evolve over time. As a result, effective computerized support of collaborative activities also must be dynamic. We see reflection as supplying a partial solution to the problem of providing dynamic computerized support to collaborative activities[1].

This paper, in particular, contains a discussion on how we perceive the use of reflection within our prototype CSCW application, ConversationBuilder (CB)[2, 7, 8], will provide dynamic support to collaborative activities. The body of this paper is structured as follows. First, we present an overview of CB including some of the theoretical foundations on which CB is based. Next, we take a brief look at reflection and the potential role reflection can play in providing dynamic support within CSCW applications. We then discuss how reflective facilities have been incorporated into CB looking primarily at two areas: process modeling and the 'language' of protocols. Included in this discussion is a brief a summary of the current state of reflection within CB as well as some future directions. Finally, we conclude with a brief summary on the role of reflection in our research.

CONVERSATIONBUILDER

ConversationBuilder is a CSCW application designed to provide *open*, *flexible*, and *active* support to collaborative activities[2]. Figure 1 gives a schematic representation of the current CB architecture. A message bus is used to connect multiple user interface suites (one per user) with the conversation engine, which provides database facilities, a hypertext system, and collaboration control[9]. New tools can be incorporated into the system by attaching them to the message bus.

Our architecture is based on a theoretical orientation we derive from the work of Winograd and Flores[16] on the equivalence of language and action. We see collabo-

[1] For more information on reflection and dynamic collaboration support systems, see the work independently carried out by Dourish[4]

[2] See [8] for more information on the nature of the open, flexible, and active support provided by ConversationBuilder.

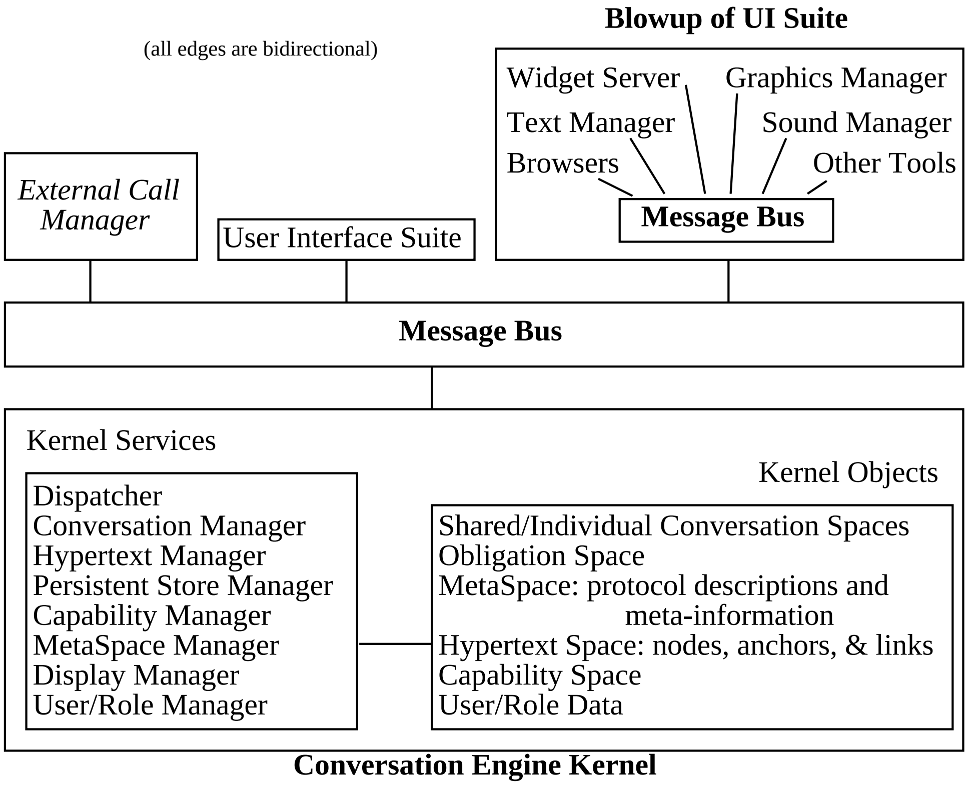

Figure 1: Architectural Schematic of ConversationBuilder

rative activities as being made up of recurrent patterns of actions (or speech acts of appropriate type), arranged such that a particular action on the part of one participant changes or opens up the possibilities for action on the part of other participants in a collaborative activity. We term such an arrangement of actions a *protocol* and any particular instantiation of a protocol a *conversation*. Therefore, a protocol specifies a family of actions, the temporal relations among the actions, and the semantics of each action, including when the action can be applied and the effect of the action (such as changing the possible actions of other participants, updating hypertext databases, and so forth). Since our approach is speech-act inspired, the protocol writer specifies the protocols using *shared* and *individual* information spaces and *utterance definitions*. The shared and individual spaces hold state information which is shared among participants in a conversation and information unique to each participant, respectively, as well as various kinds of administrative meta-information about the conversation. Utterance definitions, on the other hand, describe the actions of a collaborative activity.

CB supports simultaneous specification of multiple protocols, allowing the description of many different types of (a)synchronous collaborative activities. Furthermore, multiple instances of these protocols (i.e., conversations) are allowed to execute simultaneously.

DYNAMIC SUPPORT THROUGH REFLECTION

A reflective system has its own internal meta-structures as well as application-specific structures[3] visible to it in such a way that orderly modifications of those structures by the system are possible[14]. As a result, a reflective system is able, in effect, to modify its own behavior. More importantly from our perspective, a CSCW application which is properly reflective can, in response to appropriate stimuli[4], modify the support which it provides to collaborative activities as the needs of those activities change.

CONVERSATIONBUILDER AND REFLECTION

The process of making CB properly reflective has followed a very evolutionary path and is far from complete. However, while CB was not initially designed to be reflective, a significant amount of the meta-information

[3] The meta-structures define the behavior of the system. Application-specific structures are then built in terms of these meta-structures.

[4] Such stimuli could include: changes in collaboration participants, changes in collaboration meta-information, changes to collaboration goals, etc.

necessary to make CB properly reflective was already being explicitly maintained by the system. The progress toward making CB more reflective has moved forward on two fronts:

- explicit process modeling
- protocol language abstraction and standardization

We now take a closer look at each of these to discuss how each makes CB more reflective.

Process Modeling

Explicit process modeling was not present in early versions of CB. Instead, process models were implicitly constructed by encoding the models directly into the actions (i.e., utterances) of a protocol. As a result, it was nearly impossible for a conversation[5] to 'reason' about what effects a particular utterance had on the state of the process model supported by the conversation. Augmenting a conversation with facilities (exercised via utterances) which allow the conversation to modify its process model as the collaborative activity represented by the model changed, was equally difficult[6].

Upon recognizing this problem, we began exploring ways of making process models explicit, and therefore more easily modifiable by conversation utterances. In this setting, if we look at a conversation as being 'the program', then the flow of control of this program is established, in part[7], by its process model. Consequently, conversations which support collaborative activities that have a strong sense of *process* can 'reason' about their flow of control and modify this control during execution.

The process of experimenting with explicitly representing a conversation's process model has gone through several phases. Initially, we examined specific collaborative activities which have a strong sense of process, such as bug tracking and workflow activities, looking at ways of explicitly representing their process. From this investigation we realized that not only should a conversation's process model be visible to itself, but also to instantiations of other types of protocols. As such, we extended this idea of explicit process modeling to a more generic construct called *obligations* which extends process modeling across conversations. Obligations allow conversation process models as well as intra- and inter-conversational dependencies to be explicitly represented and subsequently built up and broken down by the system as these processes and dependencies change[8]. Although obligations are not fully implemented as designed[9], they already have proven to be an excellent testbed for future investigations into explicit process modeling of collaborative activities. Currently, we are looking at new ways of describing process models as well as more seamless integration of process models into our collaborative environment.

The 'Language' of Protocols

Making process models explicit allows the system to modify how utterances move a conversation through its process model; however, utterances do more than just effect the state and structure of the process model. Utterances may also effect conversation participants by changing participant roles and possibilities for action, conversation meta-information by modifying the shared and individual data spaces described above, and artifact information by affecting database content and hypertext structure. The dynamic nature of collaborative activities necessitates that each of these potential effects be open to modification by the system as well.

Here again, our movement towards reflection has been incremental. In earlier versions of CB, utterances were semi-structured methods[10] (i.e., code) which were shared across all conversations of a particular protocol type. As a result, it was difficult for the system to reason about utterances in a significant way because the internals of an utterance were unstructured and not directly accessible. To address the issue of how to make utterances more reflective, a cyclic process of *standardization* followed by *abstraction* was employed. This process began by standardizing the meta-information of an utterance and then abstracting that meta-information away from the 'action' the utterance performs. We call the resulting abstraction of utterance meta-information *utterance records*. Under this design, every conversation has its own set of utterance records where the meta-information contained within these records is often, but not always, shared across records. As a result, utterances, which are in a sense procedures in a program, are included as conversation data, and therefore mutable by the conversation. Currently, we are looking at finding further abstractions within utterances such that the 'action' of an utterance as well as the conditions under which the 'action' is available is more readily

[5] Remember, a conversation is an instantiation of a protocol.

[6] This could be accomplished in a very rudimentary way (one which is arguably not reflective) by augmenting every conversation with two utterances: **edit-utterance** and **recompile-utterance**. The **edit-utterance** would allow a user to edit the code that described an utterance while the **recompile-utterance** would recompile the modified code into the system.

[7] Other factors contribute to the flow of control of a conversation such as conversation participants and/or social protocol surrounding the activity.

[8] For more information about obligations see [1].

[9] Roughly 70% of the design has been implemented.

[10] By semi-structured we mean that all utterances accepted the same number of arguments and had a simple API which allowed the system either to execute an utterance in a particular context or determine if an utterance was available for execution in a particular context.

modifiable (in some cases modifiable for the first time) by the system.

CONCLUSION

Our investigations into using reflective principles for providing dynamic support to collaborative activities are very young. Primarily, we have looked at ways of structuring collaboration actions and explicitly representing collaboration process models such that they are readily modifiable from within the system. These initial efforts have shown very promising results. As our understanding of reflection and the role it can play in providing dynamic support to collaborative activities matures, we hope to further reap the potential benefits which reflection can provide to CSCW applications.

Acknowledgements

This work is supported in part by the National Science Foundation under grant CCR-8809479; by National Science Foundation/DARPA under grant CCR-9007195; by the US Army Corps of Engineers Research Laboratory; and by Intel, Bull, Open Systems Solutions Inc., and Hewlett-Packard.

References

[1] Douglas P. Bogia, William J. Tolone, Simon M. Kaplan, and Eric de la Tribouille. Supporting dynamic interdependencies among collaborative activities. In *Proceedings ACM 1993 Conference on Organizational Computer Systems (COCS)*, Milpitas, CA, November 1-4, 1993. To appear in November 1993.

[2] Alan M. Carroll. *ConversationBuilder: a Collaborative Erector Set*. PhD thesis, University of Illinois, 1992.

[3] Jeff Conklin and Michael L. Begeman. gIBIS: A hypertext tool for exploratory policy discussion. *ACM Transactions on Office Information Systems*, 6(4):303–331, October 1988.

[4] Paul Dourish. Computational reflection and CSCW design. Technical report, Rank Xerox EuroPARC, Cambridge, UK, 1992.

[5] Irene Greif, editor. *Computer-Supported Cooperative Work: a Book of Readings*. Morgan Kaufmann Publishers, Inc., 1988.

[6] Hiroshi Ishii, Minoru Kobayashi, and Jonathan Grudin. Integration of inter-personal space and shared workspace: Clearboard design and experiments. In *Proceedings ACM Conference on Computer-Supported Cooperative Work (CSCW '92)*, pages 33–42, Toronto, Canada, Oct. 31 – Nov. 4, 1992.

[7] Simon M. Kaplan, Alan M. Carroll, and Kenneth J. MacGregor. Supporting collaborative processes with ConversationBuilder. In Peter de Jong, editor, *Proceedings ACM 1991 Conference on Organizational Computer Systems (COCS)*, pages 69–79. ACM Press, 1991.

[8] Simon M. Kaplan, William J. Tolone, Douglas P. Bogia, and Celsina Bignoli. Flexible, active support for collaborative work with ConversationBuilder. In *Proceedings ACM Conference on Computer-Supported Cooperative Work (CSCW '92)*, Toronto, Canada, Oct. 31 – Nov. 4, 1992.

[9] Simon M. Kaplan, William J. Tolone, Alan M. Carroll, Douglas P. Bogia, and Celsina Bignoli. Supporting collaborative software development with ConversationBuilder. In *Proceedings of the Fifth ACM SIGSOFT Symposium on Software Development Environments*, pages 11–20, Washington, DC, December 1992.

[10] Michael J. Knister and Atul Prakash. Distedit: A distributed tookit for supporting multiple group editors. In *Proceedings of the Conference on Computer-Supported cooperative Work (CSCW '90)*, pages 343–355, Los Angeles, CA, October 7-10, 1990.

[11] Lotus. *Lotus Notes Users Guide*. Lotus Development Corp., Cambridge, MA, 1989.

[12] Iva Lu and Marilyn Mantei. Idea management in a shared drawing tool. In *Proceedings of the Second European Conference on Computer-Supported Cooperative Work*, pages 97–112, Amsterdam, The Netherlands, September 25-27 1991.

[13] Pattie Maes. *Computational Reflection*. PhD thesis, Laboratory for Artificial Intelligence, Vrije Universiteit Brussels, 1987.

[14] Pattie Maes. Concepts and experiments in computational reflection. In *Proceedings of the Conference on Object-Oriented Programming: Systems, Languages, and Applications*, 1987.

[15] Thomas W. Malone, Kum-Yew Lai, and Christopher Fry. Experiments with oval: A radically tailorable tool for cooperative work. In *Proceedings ACM Conference on Computer-Supported Cooperative Work (CSCW '92)*, pages 289–297, Toronto, Canada, Oct. 31 – Nov. 4, 1992.

[16] Terry Winograd and Fernando Flores. *Understanding Computers and Cognition*. Addison-Wesley, Reading, Mass., 1987.