Introducing Collaboration into an Application Development Environment

Susanne Hupfer, Li-Te Cheng, Steven Ross, John Patterson
IBM Research
Collaborative User Experience Group
1 Rogers Street, Cambridge, MA
{Susanne_Hupfer, Li-Te_Cheng, Steven_Ross, John_Patterson}@us.ibm.com

ABSTRACT
We present contextual collaboration, an approach to building collaborative systems that embeds collaborative capabilities into core applications, and discuss its advantages. We describe the Jazz collaborative application development environment that we are using to explore this concept and discuss design guidelines that have emerged from our experience.

Categories and Subject Descriptors
H.5.3 [**Information Interfaces and Presentation**]: Group and Organization Interfaces – *Computer-supported cooperative work*, *Collaborative computing*; D.2.6 [**Software Engineering**]: Programming Environments– *Programmer workbench, Integrated environments, Interactive environments*

General Terms
Design, Human Factors

Keywords
Computer-supported cooperative work, contextual collaboration, integrated development environment, IDE, collaborative development environment, CDE, software development

1. INTRODUCTION
To communicate and collaborate with one another, people working together on a project may deploy general-purpose collaborative systems that incorporate a set of multi-user tools (e.g. workspaces with text editors, chat utilities, whiteboards, and document repositories) [8, 9]. Alternately, they may simply continue using the specialized, single-user tools of their trade (e.g. spreadsheets, CAD tools, integrated development environments for building software) and co-opt existing communication tools such as email and instant messaging as their means of collaboration [7]. In the former approach, collaborators need to "go there" – to the team room or workplace – to work together on shared artifacts. In the latter laissez-faire approach, people "stay here" in their conventional tools – leaving them in order to do limited, ad hoc collaboration – and the artifacts end up scattered among participants' email inboxes, file systems, and other tools.

CSCW '04, November 6–10, 2004, Chicago, Illinois, USA.

A promising third approach is to bring the collaboration "into context": People continue to use their core applications, but collaborative components and capabilities are embedded within the tools. Thus, without leaving the application in which they normally work, users can communicate and collaborate with their colleagues about the project at hand. This approach is referred to as *contextual collaboration*[1] [6].

In this paper, we discuss *contextual collaboration* in more detail, present its advantages, and reference related work. We then introduce Jazz, a research project at IBM that embraces the contextual collaboration approach by embedding a set of collaborative features into the Eclipse application development environment [5]. Next we reflect on the design guidelines for contextual collaboration that we have learned through our experience with Jazz; these are meant to serve as a starting point for others who are interested in outfitting existing software with collaborative capabilities. Finally, we summarize our experiences with contextual collaboration and indicate future directions.

2. CONTEXTUAL COLLABORATION
The concept of contextual collaboration has been spearheaded by industry and covered in the trade press, but has been largely overlooked by the research community [6]. The phrase refers to an approach to collaboration in which users are not forced to leave their core applications to launch collaborative tools or visit a collaboration platform; instead, collaborative capabilities are simply available as components that extend standard applications.

Contextual collaboration has a number of advantages over other methods of supporting collaboration. Perhaps the most significant benefit is that it can **reduce friction** [1]. Embedding collaboration seamlessly into host applications spares users the time and effort of context switching to other tools whenever they need to communicate or work together, allows them to remain focused on the task at hand, and saves them the overhead of learning whole new systems. One simple but powerful example of contextual collaboration is the "Live Names" feature in IBM Lotus Workplace. Names that appear anywhere in the system (e.g. in the body of an email) serve as a launch point for looking up the person's contact information, saving it to an address book, indicating if the person is online, and initiating a chat. "Smart Tags" in Microsoft Office XP are a similar example; they could be used to retrieve information from a shared repository, for instance.

[1] The phrase was coined by Matt Cain, of Meta Group, to refer to the union of business processes and collaboration services.

Booch and Brown have discussed the value of reducing friction in the software development process, which is inherently collaborative [1]. Integrated development environments (IDEs) – where programmers carry out most of their work – have already made limited strides toward this objective by incorporating tools such as compilers, linkers, and debuggers. In some cases, IDEs integrate source code control tools (e.g. CVS, Rational ClearCase) as well; these serve as shared repositories for a software development team's code artifacts and also support structured collaboration, allowing developers to share, edit, and merge files in a coordinated way without leaving the IDE context. Booch and Brown contend that additional mechanisms for team collaboration should be brought into the development environment to further reduce friction; they advocate full-fledged collaborative development environments (CDEs) – virtual project spaces where all stakeholders can exchange knowledge, converse, brainstorm, and work together on a common task, e.g. a software deliverable.

A second benefit of contextual collaboration is that context can be used to **enhance collaborative work**. Consider the ad hoc collaboration that occurs when workers email or chat about a document they are constructing. If the conversation contains particularly useful information, a participant is likely to archive the email or save the chat transcript. But consider what happens when one later wants to retrieve the discussion. Was it an email or a chat? Who saved it, and where? Even if transcripts and emails can be located and rummaged through, the work they reference may not be obvious, because the discussions are completely decoupled from the work artifacts. Churchill et al. have described an Anchored Conversations tool that allows text-based chats to be "anchored into" the documents that are the basis of the work [4]. These contextual chats are accessible to participants from pushpin icons in the document's text – allowing users to easily locate and revisit discussions and leave messages – and also from a searchable chat database. When a user accepts a chat invitation, the associated document is automatically delivered, opened, and focused on the chat location – everyone is "on the same page." This example illustrates that contextual collaboration can enhance teamwork by establishing a shared, persistent context and by easing the collection and retrieval of collaborative artifacts.

Contextual collaboration can also better **inform collaborative work.** Consider what happens when a user initiates an anchored chat: All participants immediately know what work is being discussed; there is no need to tell them where to navigate or to paste in relevant text. Similarly, one can imagine a financial analyst initiating a chat with colleagues from within a cell of a spreadsheet; even if they are not in the same application, they can automatically be informed of the context of the call. If co-workers are using core applications that have been outfitted with contextual collaboration, those applications will know about each user's current actions – e.g. editing a certain file, debugging code, or chatting with co-workers – and can furnish that information. Better awareness of colleagues' context can forestall duplication of effort, inform whether or not to interrupt someone, and so on.

Finally, contextual collaboration lends itself to **reuse of collaborative components**. Instead of being entwined in a monolithic collaborative platform, collaborative capabilities (e.g. presence awareness, chat, application sharing) should be designed as independent, reusable components, with well-defined APIs, that can be embedded and used in any core application.

3. JAZZ: A COLLABORATIVE DEVELOPMENT ENVIRONMENT

We are gaining experience with contextual collaboration through Jazz, a research project at IBM focused on embedding collaborative capabilities into an application development environment. Booch and Brown posit that a rich collaborative development environment emerges from the collection of many apparently simple collaborative components that support coordination, collaboration, and community building – the essential "Three Cs" of CDEs [1]. They further contend that IDEs augmented with team-centric features are superior to those merely enhanced with some collaborative support. Sawyer and Guinan have studied software development and reported on the positive impact of team-level social processes on product quality and team performance [10]. Our objective with Jazz is to build a CDE that embodies the "Three C's," promotes interactions among a close-knit team of developers, and captures the team's artifacts to provide a useful knowledge base and context for communication.

Jazz is based on the metaphor of an "open office" for software development, in which a small, core team of developers works in close proximity at their workstations, with a shared space available for collaborating at whiteboards, sharing materials, or having meetings [2, 9]. Team awareness is a significant characteristic of this environment: Even while concentrating on their own work, developers have a peripheral sense of the work, activities, and discussions going on around them. Communication is another vital aspect of the open office: Team members shout out questions or information to the team as a whole, or call colleagues over to their workstation to consult on matters.

4. DESIGN GUIDELINES FOR CONTEXTUAL COLLABORATION

The initial design step for contextual collaboration is to choose an extensible infrastructure as the basis for one's work. The implementation approach we have taken with Jazz is to extend the Eclipse Java development tools (JDT) that implement a Java IDE [2, 5]. Eclipse has been designed especially for extensibility; it has a means for defining new functionality (plug-ins) and a contribution mechanism for adding new capabilities to plug-ins (extension points) [5]. Furthermore, it is an open source project, so in cases where the JDT was not extensible enough for our purposes, we were able to dig into the source code and leverage internal APIs. In this section, we will review some additional design guidelines that have emerged from our work on Jazz.

4.1 Add Collaboration Unobtrusively

Given the core application being equipped with collaboration, an initial step is to determine what collaborative capabilities to provide, and how they will appear and function in the context of that application. One guiding principle is to *reduce friction*, not increase it, so the new features should be unobtrusive and avoid interfering with the host application's expected look and functionality. Accordingly, the features *should not commandeer much space* from the application, and *should conform to its user interface metaphors* as much as possible.

Our Jazz-enhanced IDE adheres to these points. Jazz is designed to support small, informal teams; anyone can create a team and add or remove members. The Jazz UI elements abide by Eclipse's user interface metaphor of views and perspectives. A view is an

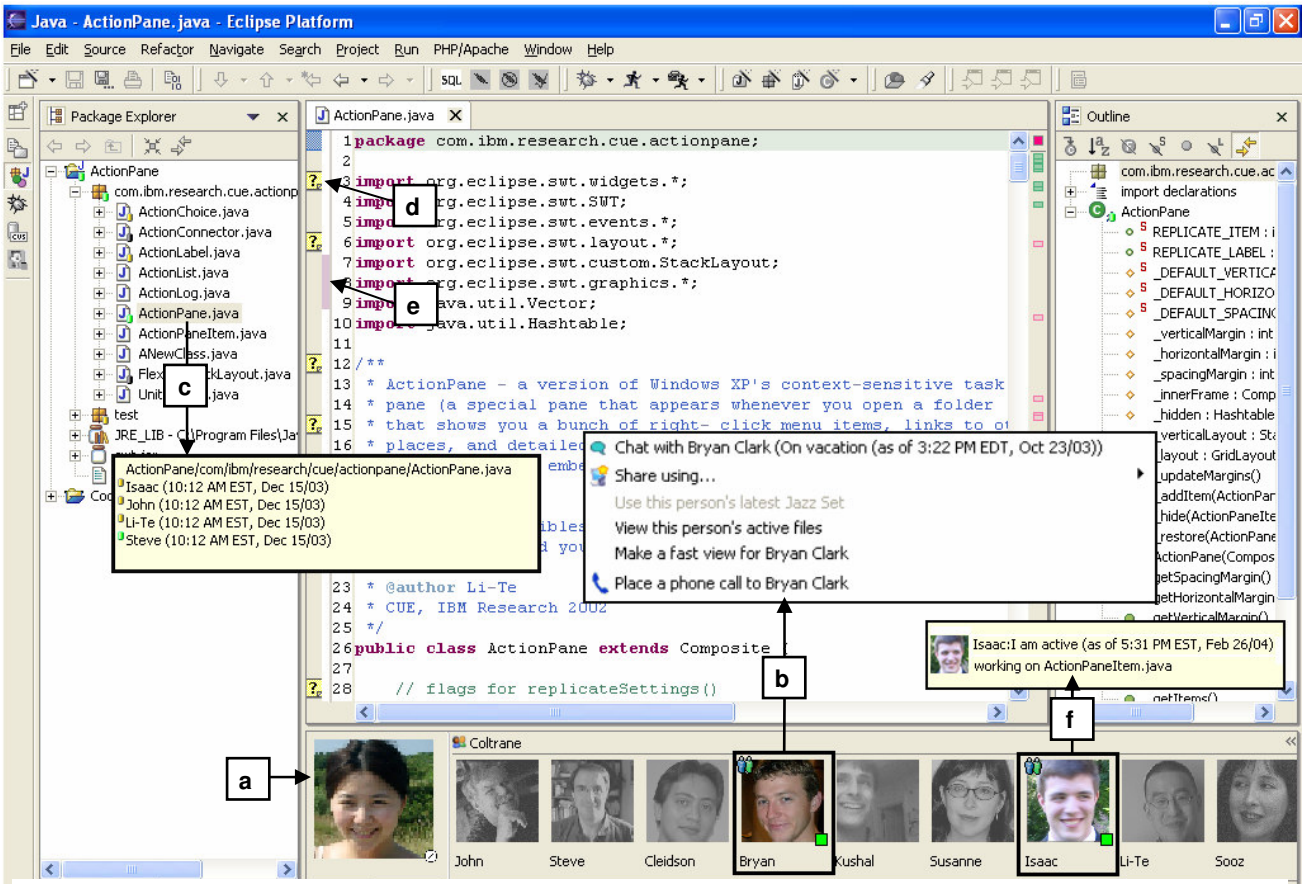

Figure 1. a) Jazz Band showing teams, members, and status icons, b) menu offering communication options, c) decorators and tooltips on resources, d) anchored chat marker, e) code modification indicator, f) team member's status message

embedded, but moveable, window, and perspectives represent an entire screen of views (e.g. the Java development perspective). Extensions like Jazz can build upon existing views and also define new views.

The most visible enhancement to Eclipse is the Jazz Band (Fig. 1a), a view that acts as a shared buddy list, displaying the teams the user belongs to and their members. Members are represented by images, decorated by status icons at the bottom right indicating whether the person is online, away, or busy; any changes to the state immediately show up on all team members' Jazz Bands. In the event the Jazz Band takes up too much space for a user's taste, it can be resized to be much thinner. The Jazz Band also serves as the launching point for a variety of interactions; right-clicking a teammate's image pops up a menu that allows one to initiate a text chat, voice-over-IP, or screen sharing session (Fig. 1b). We have reduced friction by making these interactions easily available from the IDE, without requiring any additional setup overhead (e.g. setting up servers or configuring IP addresses).

The Jazz Band is the only element we have added to Eclipse that appropriates space from the IDE. In the event a collaborative activity needs more space than we are willing to take from the IDE, we pop it up; for example chat sessions open in their own windows rather than being shoehorned into the IDE. In those rare cases when an interaction requires even more space and demands a user's full attention, we leave the Java perspective and open the collaboration in its own perspective. For instance, when team members start a screen sharing session, we take them to a special screen sharing perspective in Eclipse (though the Java perspective remains just a button-click away). As we'll see shortly, however, wherever possible we have managed to insinuate collaboration into the IDE's user interface in more subtle ways to minimize friction; we have been able to achieve this largely due to the extensibility of the Eclipse IDE and its UI elements.

4.2 Insinuate Collaboration into the Context

In addition to the people-awareness afforded by the Jazz Band, Jazz also provides resource-awareness via extensions to the IDE's Package Explorer. Files in the explorer (Fig. 1c) are decorated with colored icons to signify what teammates are currently doing with the resources (e.g. green indicates a file is currently open and being edited, yellow signals that a file has been modified locally but not checked back into the source control repository, and black denotes that a file has been checked back in). Hovering over the resource brings up a tooltip displaying who is responsible for the changes (Fig. 1c). These decorators and tooltips appear in the context where the developer regularly manages files and represent a low-friction way to inform a user in real time about the activities of other teammates on shared resources.

Another way we have insinuated collaboration into the context is through anchored chats. A developer can highlight a region of code in the editor, right-click, and initiate a chat about it; when the discussion ends, a transcript can be saved and will appear as a

marker in the left margin next to the relevant code (Fig. 1d). Teammates can later review the chat by clicking on the marker. Individuals can annotate code using the same technique. Other markers (Fig. 1e) are used to signal that a team member has modified a particular area of code; hovering over the marker shows the difference between the local code and the remote code on the teammate's desktop. These various markers are a subtle way of letting developers know where the attention of other team members is focused in the codebase and what areas of code are being discussed. Armed with this resource-awareness, developers can better coordinate their efforts and avoid conflicts.

Eclipse's extensibility and openness allowed us to achieve much of this resource-centered awareness. For example, we were able to define an Eclipse extension for file decorators and an extension for the "file differencing" feature to customize our Jazz UI. In other cases, the ability to dig into the open source code proved necessary; where Eclipse did not provide certain extensions we would have liked, we were able to locate public APIs in the source code that provided the desired functionality. At the other end of the integration spectrum, if one is attempting to retrofit collaboration into a closed application that does not support extensions or revising and recompiling the original code, the only way to insinuate collaboration into the application may be through the use of aspect-oriented programming [3].

4.3 Insinuate Context into the Collaboration

Once collaboration has been integrated into the context of a core application in unobtrusive ways, the next level of integration can be considered – insinuating context into the collaborative features. Consider the Jazz Band: If we hover over a person's image, we see a tooltip that displays their status message (Fig. 1f). Since Eclipse has access to a range of contextual information about a user's current activity in the IDE – e.g. the active project, perspective, editor, and file – developers can optionally reveal this real-time information in their status messages by using macros. Jazz also knows when various interactions – chats, screen shares, or VoIP communications – are occurring and can display that information; e.g. the decorators on the upper left of the "Bryan" and "Isaac" portraits in Fig. 1 indicate that those team members are currently in a screen sharing session. Contextual indicators in the Jazz UI are helpful in keeping team members aware of who is working on what, who is communicating, and where the action is, and they can use the indicators to decide when to contact others.

We have also been able to insinuate contextual information into our chats. When a user selects code and initiates a conversation, that code is automatically pulled into the chat window, with the proper Java formatting. Chats have been enabled to provide auto-completion and hyperlinking of team member names when entered as chat text, and right-clicking on a name brings up a menu that serves as a launching point for other possible interactions, e.g. a screen share with that person. Code fragments that are pasted or typed into the chat are also recognized and formatted properly, and filenames that are entered can act as hyperlinks taking one directly back to code modules in the editor.

5. CONCLUSIONS

We have discussed a number of benefits to the contextual collaboration approach: reducing friction, enhancing and informing collaboration, and encouraging reuse of components. Based on our experiences with building Jazz, we have presented general design guidelines for contextual collaboration that are intended to maximize its advantages.

Jazz has taken a number of steps to utilize contextual information. The project is currently exploring additional ways to support contextual collaboration, including a shared, searchable team space that logs all team events and artifacts (chat transcripts, code merges, alerts, documents, messages or questions for the team). Another research focus is interruption management: The ease of online communication raises concerns that users will be interrupted too often, but the embedding of collaboration tools in the working environment gives us the opportunity to provide a unified mechanism for managing a variety of interruption sources. Work is underway to use the contextual information available in Jazz to support more sophisticated interruption management schemes and thus further reduce the friction. Contextual collaboration has the potential to improve both the working experience and the experience of working together. Our continuing research on the Jazz project will be focused on enhancing and measuring the benefits of this approach.

6. REFERENCES

[1] Booch, G. and Brown, A.W. Collaborative Development Environments, in *Advances in Computers Vol. 59*, Academic Press, Aug. 2003.

[2] Cheng, L., Hupfer, S., Ross, S., and Patterson, J. Jazzing Up Eclipse with Collaborative Tools. In *Proc. of 2003 OOPSLA Workshop on Eclipse Technology eXchange* (Anaheim, CA, Oct. 27, 2003). ACM, New York, 2003, 45-49.

[3] Cheng, L., Rohall, S.L., Patterson, J., Ross, S. and Hupfer, S. Retrofitting Collaboration into UIs with Aspects. In *Proc. of ACM 2004 Conf. on Computer Supported Cooperative Work (*Chicago, Nov. 6-10, 2004). ACM, New York, 2004.

[4] Churchill, E.F., Trevor, J., Bly, S., Nelson, L., and Cubranic, D. Anchored Conversations: Chatting in the Context of a Document. In *Proc. of SIGCHI Conference on Human Factors in Computing Systems* (The Hague, Netherlands, April 1-6, 2000). ACM, New York, 2000, 454-461.

[5] Eclipse.org, http://www.eclipse.org

[6] Fontana, John. Collaborative Software Ages Slowly. In *Network World Fusion*, January 6, 2003.

[7] Greif, I. and Millen, D.R. Communication Trends and the On-Demand Organization. IBM Research, 2003.

[8] Ozzie, R. and O'Kelly, P. Communication, Collaboration, and Technology: Back to the Future. Groove Networks,2003.

[9] Roseman, M. and Greenberg, S. TeamRooms: Network Places for Collaboration. In *Proc. of ACM 1996 Conference on Computer Supported Cooperative Work (*Boston, MA, Nov. 16-20, 1996). ACM, New York, 1996, 325-333.

[10] Sawyer, S. and Guinan, P. Software Development: Processes and Performance. *IBM Systems Journal*, *37*, 4 (1998), 552-569.