In Proceedings of the International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom) 2009, pages 1-9, Crystal City, Washington D.C., USA, November 11-14, 2009, IEEE.

Multi-user Multi-account Interaction in Groupware Supporting Single-display Collaboration

Bastian Steinert*, Michael Grünewald†, Stefan Richter‡, Jens Lincke*, and Robert Hirschfeld*
Software Architecture Group
Hasso Plattner Institute
University of Potsdam
http://www.hpi.uni-potsdam.de/swa
*Email: {firstname.lastname}@hpi.uni-potsdam.de
†Email: michael.gruenewald@student.hpi.uni-potsdam.de
‡Email: stefanc.richter@student.hpi.uni-potsdam.de

***Abstract*—Combining support for single display collaboration with support for asynchronous and remote collaboration in one groupware challenges some basic assumptions of application design and brings up new requirements for application platforms. While user accounts are central in many kinds of groupware, they are not respected in groupware support for multi-user single-screen interaction. Current support for this interaction paradigm does not allow users to act on their own behalf; they have to act on behalf of a host user.**

We suggest an approach to distinguish the interactions with different users in multi-user single-screen scenarios. Our approach enables applications to link actions to the acting user's account. We describe the integration of suggested concepts in the groupware ProjectTalk, an application for managing XP projects that supports multi-user single-screen interaction. All interacting users are allowed to work with ProjectTalk on their own behalf.

I. Introduction

User accounts are central to many kinds of groupware, for example, to asynchronous groupware such as Wikis [1] or project management applications. These kinds of groupware usually support collaboration amongst individuals contributing at different times and from different places [2], [3]. Typically users are required to log in first—enabling personalization of content, access restriction, or tracing a user's activities. However, these groupware system rely on single-user single-display interaction, that is, application clients are used by one user at a time.

This assumption does not hold true for Single Display Groupware (SDG) [4] or screen sharing technologies such as Virtual Network Computing (VNC) [5]. The former supports close collaboration of co-present users by allowing them to interact with an application on a single display simultaneously. The latter allows remote users to look at the same screen from another computer desktop. In the following, the term *multi-user single-screen interaction* refers to interaction scenarios where multiple users interact with the same screen, shared either locally or remotely.

We gratefully acknowledge the financial support of the Hasso Plattner Design Thinking Research Program for our project "Agile Software Development in Virtual Collaboration Environments".

Combining support for single display collaboration with support for asynchronous and remote collaboration in one groupware challenges some basic assumptions of application design and brings up new requirements for application platforms. While many groupware systems typically build on a user account concept, current approaches to support multi-user single-screen interaction do not allow users to act on their own behalf. Users rather act on behalf of the user who logged on before. As one important consequence, users are not allowed to perform the same set of actions as they would be when acting on their own behalf.

In this paper, we suggest an approach for supporting both paradigms adequately. We present concepts to distinguish actions of different users and describe how these actions can be linked to the acting user's account. The suggested approach was successfully employed in a groupware called ProjectTalk illustrated in Fig. 1, which supports both synchronous and asynchronous collaboration scenarios. We designed and de-

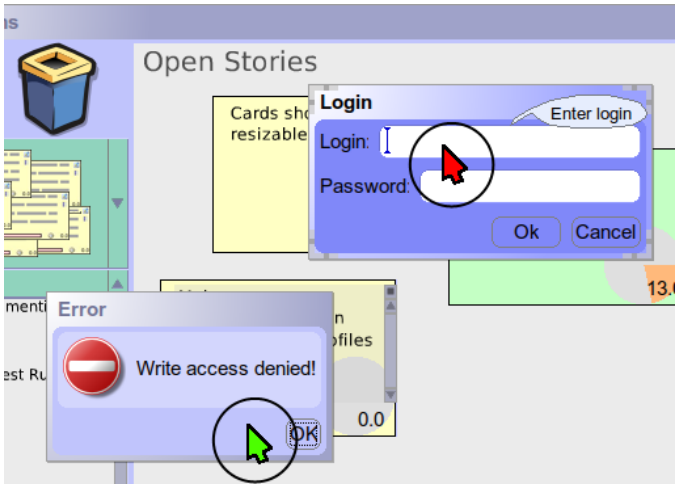

Figure 1. Two users are interacting with ProjectTalk where each of them is represented by a mouse cursor. One of the users (right mouse cursor) is requested to initially provide username and password. The other user (left mouse cursor) is informed about missing privileges to change a user story.

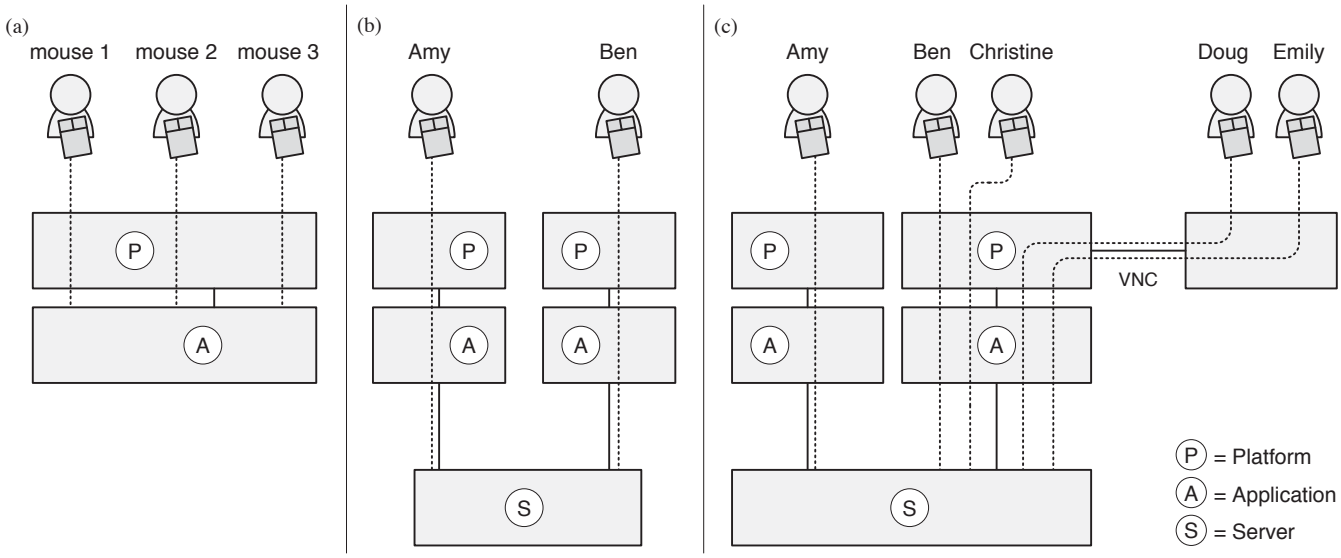

Figure 2. Different concepts of multi-user software: (a) In classic SDG applications, multiple input devices are available but without user details; (b) in client-server applications, user information are available to the server but users cannot share an application instance; (c) with our approach user information are transported from input devices to the server.

veloped ProjectTalk to support team activities of Extreme Programming (XP) projects [6] such as writing user stories or planning iterations. As these activities require close cooperation of all participants, our application supports multi-user single-screen interaction. Remote team members can join ProjectTalk sessions by using screen sharing technology. In ProjectTalk, all collaborators either co-present or remote can act on their own behalf.

Section II describes ProjectTalk as a sample groupware supporting multi-user single-screen interaction. By means of covered scenarios, we describe limitations of current approaches to support multi-user single-screen interaction. In Section III, we introduce our approach to distinguish interactions with different users and present required extensions to the Squeak application platform. Section IV describes how actions can be linked to the acting user and how this approach is integrated into ProjectTalk. In Section V, we discuss existing SDG frameworks and related remote collaboration approaches.

II. Merging Characteristics of Asynchronous Groupware and Single Display Groupware

ProjectTalk is an asynchronous groupware (Fig. 2b,c) that we designed and implemented in the course of our research activities. It facilitates collaborative activities in XP teams. To enable spontaneous interaction, ProjectTalk was also designed to support single-display multi-user interaction (Fig. 2a,c). The application was built in Squeak Smalltalk utilizing Morphic as the standard GUI framework.

A. ProjectTalk: A Case Study

Close collaboration and communication among team members and the customer is vital In XP projects. All team members and the customer regularly reflect on the project's status and plan future releases. Participants discuss feature requests and develop a common understanding of the required functionality.

Traditionally XP teams heavily rely on physical tools for communication and organization like index cards, whiteboards and post-it notes. Their main tool is the user story card, a small index card with a short, textual description of an application feature. These cards are pinned on a whiteboard and form the basis for a common understanding. Bringing all these information from the physical whiteboard into the digital world would enable a multitude of new possibilities, such as having unbounded space or support for full text search. Additionally it enables remote collaboration, because virtual whiteboards, unlike physical ones, can be shared over computer networks.

By employing SDG concepts, we provide XP teams with interaction characteristics similar to working with physical tools (Fig. 3). By using physical tools such as whiteboards and index cards, team members are able to act independently of one another without the need to synchronize on pens, for example. Traditional applications, however, only support interaction with one user at a time. The need to synchronize on application control impedes spontaneous interaction and reduces social dynamic in comparison to a physical whiteboard. Therefore we enable multiple users to interact with the application independently. To further increase the dynamic of a session, users are able to join or leave a session at any time.

We also incorporated screen sharing technology to support distributed XP teams (Fig. 2c). Remote team members can share planning sessions, for example, and interact with the same shared screen. Screen sharing technology such as VNC is a simple approach to support synchronous remote collaboration. Still, it allows participants to see others interacting and does not require modifications to the application.

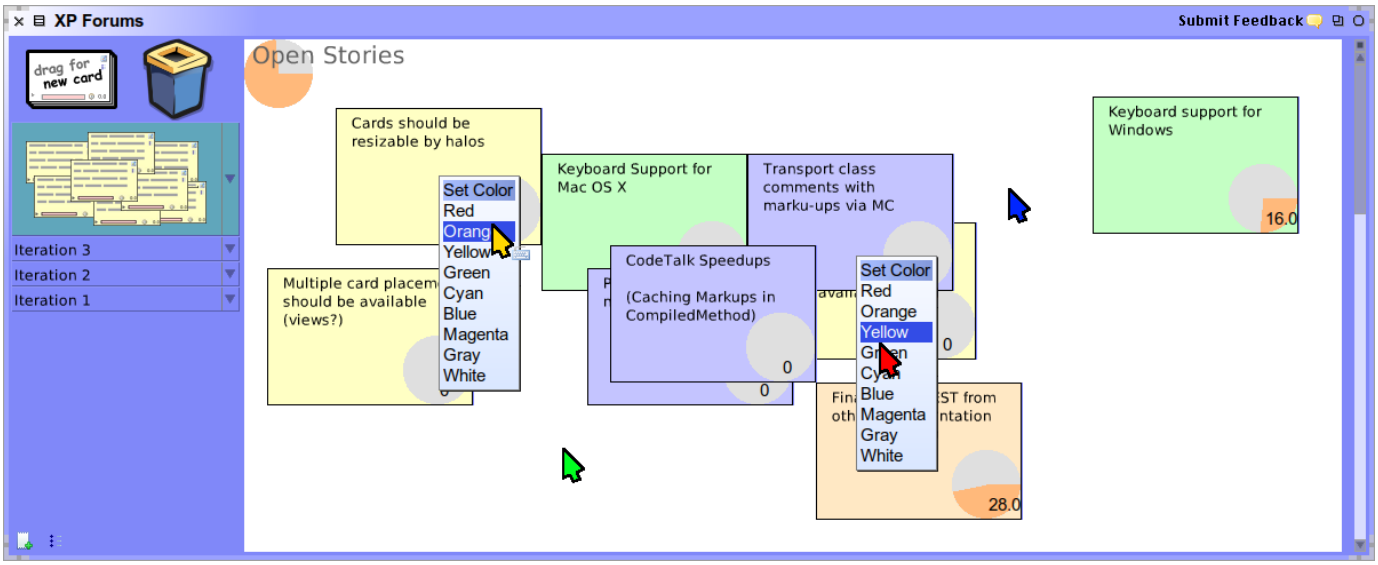

Figure 3. Screenshot of the application: Multiple users, represented on the screen by colored mouse cursors, interact with a virtual whiteboard. Every user may open its own context menu.

B. Issues concerning User Accounts

Traditional applications assume that only one user interacts with one computer at a time. This assumption results in well-know login screens requiring users to log in before they can use provided functionality. After having logged in, all further actions of users are associated with their respective digital identity.

When multiple users interact with a single screen, the application is unable to distinguish acting users. These restrictions of current concepts lead to issues concerning authorization and traceability. Fig. 4 depicts a typical multi-user single-screen scenario. In this scenario, the application is unable to make a reasonable decision whether the user is privileged to perform the desired action as the application does not know who the the currently acting user is. By using current approaches, all users actually act on behalf of a host user, the one who logged in before. This gives all acting users the same privileges in the described scenario. The inability to distinguish multiple acting users does not only lead to undesired modifications, but also to unintended restrictions of users. When, for example, users want access to previous projects for analysis purposes, they might be unable to open the projects as the host user is not privileged for accessing this information. As another consequence of application actions not being linked to the acting user, tracing data of user action become unreliable. It is impossible to find out who modified certain important information.

The examples described above show that it is necessary to distinguish users concerning their security context and to execute every action users want to perform in their respective context. In particular, the following questions come up:

- How might an application be designed to allow multiple interacting users logging in and logging out?
- How should the user credentials be managed?
- How might UI events be distinguished by acting users?
- How might the application make use of this distinction and link application actions to users?
- How should applications be designed to deal with multiple interacting users having different privileges?

III. Platform Support for Multi-User Multi-Account Interaction

An application featuring multi-user single-screen interaction requires special support in the application's platform such as handling the events from multiple, similar input devices independently from each other. In addition, if multiple remote users should be able to work in same way as local users, and if UI actions should be linked to the users, an adequate concept representing the users and their actions is needed.

A. Using Hands to Represent User Input

Morphic [7] is a completely object-oriented GUI framework. First implemented for the Self programming language, it was later adopted by Squeak Smalltalk. In applications implemented with Morphic, every visible element on the screen is represented by an object in the code, even the mouse pointer. These objects used by Morphic are called *Morphs*, according to this the object representing the mouse pointer is called *HandMorph*. The concept of hands allows having multiple hands that can be controlled by different sources.

Every hand object is responsible for communication with the respective input source and generates corresponding UI framework events. The default hand communicates with the operating system and processes default streams of mouse and keyboard events. Hand object are also able to determine the UI element (the *Morph*) that is currently focused (keyboard event) or that is directly below the mouse pointer. With that knowledge they trigger the corresponding application event processing for each incoming event.

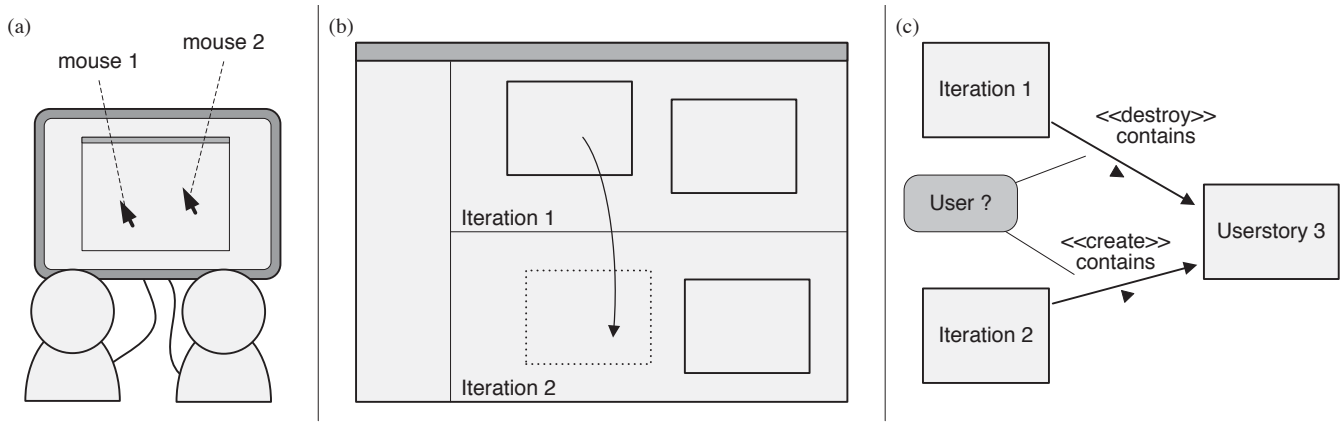

Figure 4. A participant moves a user story from one iteration to another. Due to the missing user context, the application cannot determine the user who initiated the action, and is unable to check whether the acting user is privileged to perform the desired action.

B. Controlling Hands via Multiple Input Devices or VNC

When multiple users interact with a single-screen, it is usually desired that they can work with the application independently. This is comparable to a physical whiteboard, where multiple users can draw at the same time. This requirement can be achieved by using the concept of hands. Hand objects obtain their event data from the corresponding input stream autonomously. By using this concept, additional input streams can be integrated; multiple hands, that is, mouse pointers and cursors, can be controlled by different input source.

Common operating systems can activate multiple input devices, but support only one system cursor. Input events from these different devices are merged into one input event stream. Thus, it makes no difference whether the connected mouse or the built-in touchpad is used to control the system cursor. To bypass the operation system's behavior, we developed special support for Squeak's virtual machine (VM). The VM plugins access separate streams of input event data using special lower-level operating system APIs. Details of this approach are described in [8], [9]. The design of our VM extensions allows attaching and detaching input devices during the application's run-time.

Screen sharing or terminal server technologies, such as VNC or the Remote Desktop Protocol (RDP), also support only one cursor per session over all participants. The applications and underlying protocols expect only input stream of one user. In addition, the host user and the remote user share control over the application using a single mouse pointer and cursor. The selected application platform Squeak Smalltalk already has integrated screen sharing support providing a VNC viewer and server. We extended Squeak's VNC implementation, so that client-side input events of different hands are transmitted to the server in separate conceptual event streams. The platform sharing the application manages dedicated hand objects for the different event streams.

Hand objects abstract from concrete input sources and provide a defined interface to applications. Thus, it is transparent to the application, whether they are controlled from a local device or via a VNC connection.

C. Impersonating Hands

Until now multiple local or remote users can interact with the application simultaneously and independently, because users have their own hands. However, there is no relation between the user and a cursor on the screen, that is, the application does not have any information about the source of events.

To provide this link between users and their hand, we suggest to impersonate hands. Subsection III-A describes that hand objects trigger the application's processing of UI framework events. We argue that every event processing should be performed on behalf of the corresponding users. As we describe below, hand objects are therefore required to manage digital identities of acting users.

According to the suggested concept, hand objects have to manage user credentials. When, for example, a new input device is attached, a corresponding hand is created, not yet providing any credentials. When the applications requires credentials of the acting user, it has to ask the acting hand for this information. Based on that request, the hand object opens a dialog asking the user to provide username and password. This information is then stored in the hand object for further use. To avoid misuse of the dialog, only the user who wants to perform the action can interact with the dialog. After completing the login, the hand represents the user and every action performed by this certain hand can be performed on behalf of the user. The next section describes how an application can use this information to link application actions to user accounts. When a user logs out, the user's credentials are removed from the hand.

When remote users connected via VNC also work with the Squeak platform, their credentials stored in the their local hand can be reused in the replicated hand. The hand morphs replicating a remote user's hand subtype the standard hand morphs. This subtype has a special behavior and can reuse credentials transmitted via VNC, which is supported by our extension to the VNC protocol. If users prefer VNC clients

that do not support our extension to the VNC protocol, the implementation will fall back to the dialog-based query.

IV. Application Support for Multi-user Multi-account Interaction

The section above introduces an extension to the hand concept for representing individual users in a groupware supporting multi-user single-screen interaction. Users can dynamically share and leave collaboration sessions; they can log in and log out. A hand may represent either co-present users or remote users connected via screen-sharing. With that, the application platform provides the prerequisites to enable users to act on their own behalf in targeted collaboration scenarios.

An application such as ProjectTalk can use the platform's capabilities so that actions can be performed in the name of the corresponding user. With that, users are able, for example, to access all data they are allowed to. The application has to link actions to user accounts, which is described in this section. This results, however, in additional authorization issues. We describe these issues and our approach to handle them. To ease understanding of descriptions, we introduce the overall design of ProjectTalk first.

A. Design of ProjectTalk

ProjectTalk is a client-server application. Similar to other collaborative client-server applications such as wikis, ProjectTalk has a central server component that allows different members of a team to contribute at different places and times. A single server handles the request of multiple clients. As all data merges at the server, it is typically also responsible for authenticating users and checking their access rights.

The client side of ProjectTalk consists basically of two conceptional layers: a view layer and a model proxy layer. Different views are bound to proxy objects in the model layer. They get notified in case of updates leading them to request up-to-date data and to render them again. The model proxy layer's main concern is communicating with the server part of our application and synchronizing all modifications to the model proxy objects. Additionally, the model proxy layer provides a repository of the root objects as the main entry point for view specific application code. Starting from this, view code has access to all objects of the model.

The client requests the server for data and is responsible for keeping requested data in sync with the server. A user action that modifies data results in request/response communication between client and server. For example, changing a user story's description (view object) results in a change of the story object's *description* property (model object) which in turn results in an update request to the server. The left side of Fig. 5 depicts involved objects and their collaboration.

The server is designed to process every request on behalf of a certain user. A client sending a request has to provide the required information about this user. Similar to other server applications, the request processing chain tries to answer three, security-related questions at first.

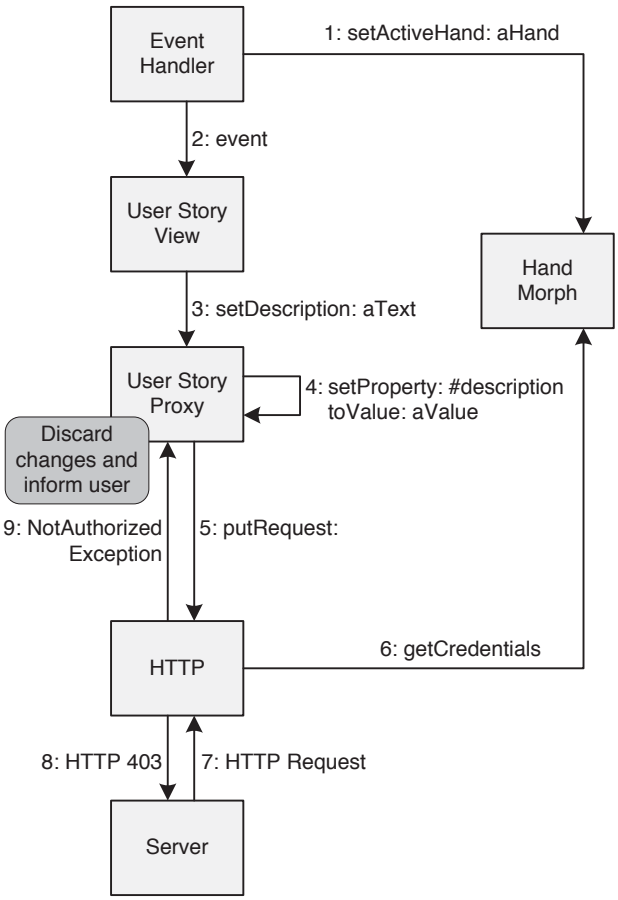

Figure 5. An exemplary processing of an event where the acting user has not the necessary privileges for the action

- Who is the user that initiated the action to be performed?
- Is the user the one he pretends to be?
- Is the user allowed to perform the action?

If provided user credentials are valid and the user has sufficient privileges, the requested actions will be performed. Otherwise, the server will respond with an appropriate error message.

B. Linking Users to Actions

The ProjectTalk client is required to provide the credentials of the corresponding user for each server request. However, a ProjectTalk client potentially interacts with many users simultaneously. Consequently, the client must provide information about the user that initiated the currently processed action. This requires the ability to distinguish performed actions by users.

The platform concepts provided so far are not sufficient to link application actions to users. A hand represents a user interacting with the application and stores the user's credentials. Interacting with a graphical user interface (GUI) results in framework events such as *MouseUp* that are handled by the application. These events are linked with the corresponding hand. So, an application method handling a GUI framework

event has access to the corresponding user information. But, an event handler method typically uses other objects of the application that do not have access to the original GUI event. For example, the handling of a *MouseUp* event, originated by a button confirming changes of a text edit field, can result in a change of a model object, updating the description of a story object (as depicted in Fig. 5).

Thus, the application requires a general concept of accessing information about the acting user. In ProjectTalk, for instance, the HTTP communication component is required to provide the acting user. When a model object in the client gets changed, the client sends an update request to the server; the HTTP component augments the request by putting the user credentials in the corresponding request header.

To our knowledge, there are three basic possibilities to provide all parts of an application a general way to access the information about the *current user* information. *Current user* refers to the user that initiated the action leading to the execution of the program fragment interested in these user data.

- *Passing an additional parameter around.* Information about the current user can be passed as an additional parameter through the application, starting from the framework providing hooks for applications to the code that wants to access these data. Thus, developers have to maintain the additional parameter for many methods although these methods are not related to user specifics.
- *Thread-local storage.* Application servers typically provide the current user to application via thread-local storage. For each incoming request, they pick a thread out of a pool. This thread may than process the request concurrently. At the beginning of the request processing, the user is stored in thread-specific storage. Application logic can access this data where ever necessary.
- *Global variables.* Global variables are another option to share information between application parts independent of message flow and parameter passing. It is, however, an inadequate option if multiple threads are executed concurrently accessing the same variable.

Other approaches, such as dynamic variables, have characteristic similar to those listed above and are thus not discussed separately.

In ProjectTalk, the global variable *ActiveHand* is used to share the current user among the application. As depicted in Fig. 5, the event handler of the GUI framework puts the hand that signaled the current event in the *ActiveHand* variable. The use of this global variable does not cause problems, because Squeak's GUI framework Morphic is single-threaded, as most GUI frameworks are. If concurrency is required, for instance, to realize asynchronous client-server communication, the user information has to be handed over to additional threads explicitly.

By accessing *ActiveHand* and putting the user credentials in the HTTP request header, ProjectTalk links all actions to users and enables them to thereby perform all actions on their own behalf.

C. Handling Client-side Authorization Issues

In groupware like ProjectTalk that supports both synchronous and asynchronous collaboration, additional authorization issues occur on the client-side. They need to be handled by the application explicitly. We discussed that many groupware systems typically build upon a user concept. When a groupware provides role-specific behavior, it is often designed in a way that users see only the functionality they have access to. In a project management software, for example, only administrative users are able to create new projects; users without these privileges cannot see this functionality. If multiple users interact with a single screen at the same time, the different users might have different privileges.

Groupware supporting multi-user single-screen can handle this mismatch in three different ways. One way is displaying the collective set of functionality all users have access to. Unfortunately, users would be limited and could not use all features they are allowed to. Another approach is presenting every feature available to at least one user. As a result, users can activate actions they are not allowed to perform. Applications have to handle denied access explicitly and be able to recover from this error. The last possibility is to (re-)design the application so that, for instance, users are provided with menus specific to their privileges.

The application design of ProjectTalk combines the second and the last option. For example, some users will not be allowed to modify user stories or move them between iterations. Still, all users have read access to these user interface components; users unprivileged to perform a modification will experience a failure and receive a corresponding message, as presented in Fig. 1. The menu for opening projects exemplifies the last option. For each user accessing the menu, it provides specific content—only projects the user is a member of.

The handling of denied access gained special attention in ProjectTalk, so that this concern is not scattered over the entire application code. A typical user interface action results in one or more HTTP request to the server. If the acting user is not authorized to read or write specified resources, the server will return with an unauthorized error. In the example depicted in Fig. 5, the user tries to change the description of a user story card. A HTTP error is responded and converted into an application specific *NotAuthorized* exception. The exception is handled in the implementation of the model proxy objects. The proxy objects provide application specific interfaces that are implemented generically. The proxy objects store object properties and synchronize modifications with the server. If the server processes the request successfully, the modification will be applied to the corresponding description property of the user story's proxy object, and bound views will get notified about the change. Otherwise, the modifications are discarded, the stories will show the old description, and the user will receive an error message.

The implementation of both handling denied access and linking actions to users is integrated well in the design of ProjectTalk. Both concerns are well-separated from application

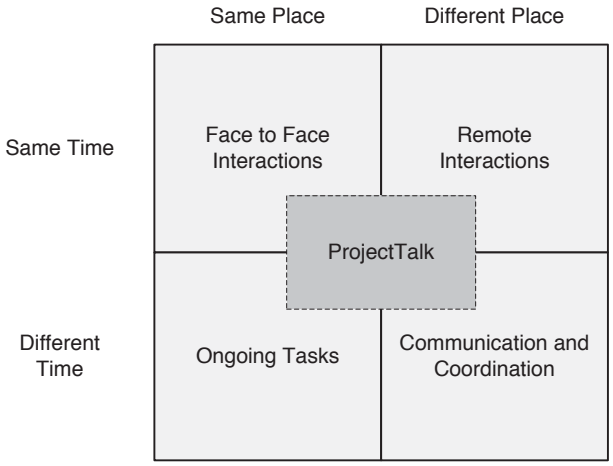

Figure 6. Classification of ProjectTalk in the *computer supported cooperative work* matrix.

specifics and the extensions can be integrated easily into other applications.

V. Related Work

The term groupware generally refers to a broad set of applications, with different kinds supporting different collaboration scenarios. The *computer supported cooperative work* matrix shown in Fig. 6 provides a classification for groupware solutions according to the context of use; the matrix was originally introduced in [3]. Following the argumentation in [2], most groupware solutions do not belong to exactly one category or another because most work activities do not strictly adhere to this classification. ProjectTalk, for example, belongs to all four categories as it supports asynchronous and synchronous collaboration scenarios, and allows co-present as well as geographically dispersed team members to contribute.

While many groupware solutions support more than one collaboration scenario, according to our knowledge, previous research has not yet addressed the combination of multi-user single-screen interaction with other collaboration scenarios. Groupware with a focus on asynchronous collaboration is widely used and well-known, but it relies on the assumption that only one user works with one client at a time. This kind of groupware typically follows the classical client-server architecture. Since many years researchers have investigated into groupware solutions that support synchronous remote collaboration. Several different approaches were developed, some are built upon a centralized and others upon a replicated architecture [10]. However, none of the current approaches meet the needs of multi-user single-screen collaboration. Benefits of using a single display by a co-present group simultaneously were, for example, described in [11].

A. Single Display Groupware

Reference [4] introduces the term *Single Display Groupware* referring to applications that allow co-present users to contribute simultaneously. This and other works [11]–[14] also provide empirical validation that working together on a task has positive effects concerning motivation and efficiency. SDG concepts become more important due to recent developments in multi-touch displays [15], in Microsoft Surface or DiamondTouch [16].

One of the first SDG applications developed is the Multi-Device Multi-User Multi-Editor (MMM) [17]. It presents a platform for different kinds of editors, for example a text editor facilitating co-present multi-user interaction. MMM already features dynamically joining a session and the authors present a first analysis of user interface design techniques.

To simplify the development of SDG applications, researchers have developed application frameworks that support processing of events from multiple input devices. The first attempt to provide such a framework was *Multiple Input Devices* [18] for the Java platform. It utilizes Microsoft DirectInput to gather data from connected mice and provides these data as Java events. However, DirectInput no longer supports some of the used functionality in Microsoft Windows 2000 and later versions. In reference [9], the authors present the SDG Toolkit for developing Microsoft .NET Framework applications. The SDG Toolkit, which is built on the Windows Raw Input interface, was the first application framework supporting separate input and control by multiple keyboards. It further provides developers with special SDG user controls forming the building blocks for new applications.

Reference [8] presents an extension to the X Window System that enables single-display collaboration in control room scenarios. Up to seven users can interact with a shared X11 screen acting from there own workstation, but only one user at a time. The authors of [19] describe the design and implementation of Multi-Pointer X Server (MPX). It allows multiple users to control different applications on the same physical screen simultaneously. All users are provided with their own mouse pointer.

However, none of the reported approaches to multi-user single-screen interaction has been combined with support for other collaboration scenarios. Current SDG solutions and technological concepts do not allow users to act on their on behalf as they distinguish only input from different devices; applications have not been prepared for linking actions to users.

B. Groupware Supporting Remote Collaboration

When the primary purpose of a groupware is asynchronous collaboration, such as in Wikis and project management applications, any kind of synchronous remote collaboration is usually not well supported. The need for remote collaboration in general, is addressed by many groupware technologies. Some of them were also specifically designed for synchronous remote collaboration scenarios. Such groupware systems typically enable awareness of collaborators interacting with the application at the same time.

To collaborate with remote sites synchronously using unprepared applications, screen sharing technologies are commonly

used, often in connection with audio or video conferencing. Virtual Collaboration Environments [20] such as such as Open Croquet [21] leverage screen sharing technology for integrating ordinary desktop applications and allow remote participants to interact with them. However, only one user can interact with the shared application at a time, and all users act on the account of the host user. The concepts suggested in this paper remove these restrictions. Participants can interact with the shared application simultaneously and are also allowed to act on their own behalf.

Apart from screen sharing, many other approaches to synchronous remote collaboration have been investigated. In contrast to centralized screen sharing setups, other approaches are based on replicated architectures where low level [22] or high level events [23] are exchanged between all participants and executed on every participant's host. References [24], [25] present frameworks that simplify the development of replicated groupware for synchronous remote collaboration scenarios. The framework described in [26] eases constructing groupware that support synchronous as well as asynchronous collaboration. However, reported approaches to supporting remote collaboration have not yet incorporated concepts of multi-user single-screen interaction.

Problems discussed in this paper are to be expected whenever a groupware that supports different collaboration scenarios such as asynchronous and synchronous remote collaboration, and that also relies on user accounts, should incorporate multi-user single-screen interaction concepts. We argue that the concepts suggested here are an adequate answer to these problems. All groupware applications that do not support multi-user single-screen interaction explicitly, rely on the assumption that only one user interacts with a workstation at a time. As a consequence, the software is unable to distinguish between multiple acting users. Extensions to current concepts as we suggest in this paper are required to enable applications to link actions and users.

VI. Summary and Outlook

In this paper, we argue for extending concepts to support multi-user single-screen interaction in groupware applications. Limitations of current concepts are described. Applications are unable to distinguish actions of different users, which, in turn, prevent users from acting on their on behalf. By using our groupware ProjectTalk, we describe different scenarios that require the distinction of acting users.

We present our extension to Morphic's hand concept that allows to distinguish UI events by acting users. We describe how applications can make use of this distinction and link application actions to user accounts. The suggested concepts unify the interaction with remote and local users as the interfaces can abstract from concrete input sources. Our approach to impersonate hands further allows to manage the credentials of all users interacting with the same screen.

The suggested concepts are integrated into ProjectTalk, a groupware we developed to support collaborative activities in XP teams. All co-present team members can interact with a single screen simultaneously. Groups of remote team members can join via screen sharing technology. We extended the VNC protocol enabling remote users to interact with the same shared screen independently. Our approach allows co-present and remote users to interact with ProjectTalk on their own behalf. The integration of our concepts into the design of ProjectTalk are described.

We also discuss issues that emerge when multiple users with different privileges interact with the same screen. We present multiple design options to handle these issues and describe our solution for ProjectTalk.

For future work, we plan to revise user interface concepts for our approach to multi-user multi-account single-screen interaction. We will investigate into providing a new kind of feedback to the users concerning their privileges. For example, it would be handy if users know whether they are allowed to perform a certain action before triggering it.

The concepts suggested in this paper allow distinguishing events of different users and thus enable applications to link actions to the acting user's account. Our presented approach to multi-user multi-account single-screen interaction allows users to interact with the same screen on their own behalf. By integrating the concepts in ProjectTalk, all users are now able to perform the set of actions they are privileged for, in particular in multi-user single-screen collaboration scenarios.

References

[1] B. Leuf and W. Cunningham, *The Wiki way: quick collaboration on the Web*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2001.

[2] J. Grudin, "Computer-supported cooperative work: History and focus," *Computer*, vol. 27, no. 5, pp. 19–26, 1994.

[3] R. Johansen, *Groupware: Computer support for business teams*. The Free Press New York, NY, USA, 1988.

[4] J. Stewart, B. B. Bederson, and A. Druin, "Single display groupware: a model for co-present collaboration," in *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA: ACM, 1999, pp. 286–293.

[5] T. Richardson, Q. Stafford-fraser, K. R. Wood, and A. Hopper, "Virtual network computing," *IEEE Internet Computing*, vol. 2, pp. 33–38, 1998.

[6] K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley Professional, 2004.

[7] J. H. Maloney and R. B. Smith, "Directness and liveness in the morphic user interface construction environment," in *UIST '95: Proceedings of the 8th annual ACM symposium on User interface and software technology*. New York, NY, USA: ACM, 1995, pp. 21–28.

[8] G. Wallace, P. Bi, K. Li, and O. Anshus, "A multi-cursor x window manager supporting control room collaboration," Princeton University, Computer Science, Technical Report TR-707-04, Tech. Rep., 2004.

[9] E. Tse and S. Greenberg, "Rapidly prototyping single display groupware through the sdgtoolkit," in *AUIC '04: Proceedings of the fifth conference on Australasian user interface*. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2004, pp. 101–110.

[10] J. M. A. Begole, "Flexible collaboration transparency: Supporting worker independence in replicated Application-Sharing systems," Ph.D. dissertation, Department of Computer Science, Virginia Polytechnic Inst. and State Univ., Blacksburg, VA., 1998.

[11] J. Stewart, E. M. Raybourn, B. Bederson, and A. Druin, "When two hands are better than one: enhancing collaboration using single display groupware," in *CHI '98: CHI 98 conference summary on Human factors in computing systems*. New York, NY, USA: ACM, 1998, pp. 287–288. [Online]. Available: http://doi.acm.org/10.1145/286498.286766

[12] K. I. Joanna, J. Mcgrenere, K. S. Booth, and M. Klawe, "The effect of turn-taking protocols on children's learning in mouse-driven collaborative environments," 1997.

[13] L. J. Bricker, L. J. Bricker, and L. J. Bricker, "Cooperatively controlled objects in support of collaboration," in *in Multimedia Applications, in Extended Abstracts of CHI'97 (Atlanta GA*. ACM Press, 1998, pp. 313–314.

[14] E. R. Pedersen, K. McCall, T. P. Moran, and F. G. Halasz, "Tivoli: an electronic whiteboard for informal workgroup meetings," in *CHI '93: Proceedings of the INTERACT '93 and CHI '93 conference on Human factors in computing systems*. New York, NY, USA: ACM, 1993, pp. 391–398.

[15] M. Morris, A. Cassanego, A. Paepcke, T. Winograd, A. Piper, and A. Huang, "Mediating group dynamics through tabletop interface design," *IEEE Computer Graphics and Applications*, vol. 26, no. 5, pp. 65–73, 2006.

[16] P. Dietz and D. Leigh, "Diamondtouch: a multi-user touch technology," in *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology*. New York, NY, USA: ACM, 2001, pp. 219–226.

[17] E. A. Bier, S. Freeman, and K. Pier, "Mmm: The multi-device multi-user multi-editor," in *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA: ACM, 1992, pp. 645–646.

[18] J. Hourcade and B. Bederson, "Architecture and implementation of a java package for multiple input devices (mid)," HCIL, Tech. Rep., 1999.

[19] P. Hutterer and B. H. Thomas, "Groupware support in the windowing system," in *AUIC '07: Proceedings of the eight Australasian conference on User interface*. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2007, pp. 39–46.

[20] S. Benford and L. Fahlén, "A spatial model of interaction in large virtual environments," in *ECSCW'93: Proceedings of the third conference on European Conference on Computer-Supported Cooperative Work*. Norwell, MA, USA: Kluwer Academic Publishers, 1993, pp. 109–124.

[21] D. A. Smith, A. Kay, A. Raab, and D. P. Reed, "Croquet - a collaboration system architecture," *Creating, Connecting and Collaborating through Computing, International Conference on*, vol. 0, p. 2, 2003.

[22] J. Begole, M. B. Rosson, and C. A. Shaffer, "Flexible collaboration transparency: supporting worker independence in replicated application-sharing systems," *ACM Trans. Comput.-Hum. Interact.*, vol. 6, no. 2, pp. 95–132, 1999.

[23] D. Li and R. Li, "Transparent sharing and interoperation of heterogeneous single-user applications," in *CSCW '02: Proceedings of the 2002 ACM conference on Computer supported cooperative work*. New York, NY, USA: ACM, 2002, pp. 246–255.

[24] M. Roseman and S. Greenberg, "Building real-time groupware with GroupKit, a groupware toolkit," *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 3, no. 1, pp. 66–106, 1996.

[25] A. Prakash and H. Shim, "DistView: Support for building efficient collaborative applications using replicated objects," in *Proceedings of the 1994 ACM conference on Computer supported cooperative work*. ACM New York, NY, USA, 1994, pp. 153–164.

[26] P. Dewan and R. Choudhary, "A high-level and flexible framework for implementing multiuser user interfaces," *ACM Transactions on Information Systems*, vol. 10, no. 4, pp. 345–380, 1992.