Introducing Collaboration in Single-user Applications through the Centralized Control Architecture

Ilaria Manno*, Furio Belgiorno*, Delfina Malandrino*, Giuseppina Palmieri*, Donato Pirozzi*, Vittorio Scarano*
*ISISLab, Dip. di Informatica ed Applicazioni "R.M. Capocelli"
Università di Salerno, Fisciano (SA), 84084, Italy
Email: {furbel,delmal,manno,palmieri,vitsca}@dia.unisa.it

***Abstract*—In this paper we describe a novel Model-View-Controller based architecture, Centralized Control, that introduces collaboration in single-users applications. The architecture is able to add collaboration with no need to modify the source code of the original single-user application, and providing also the capability to introduce group semantics into the new, collaborative application that is obtained. The architecture is shown in practice, by introducing CollabXMind, a collaborative mind map tool, that is based on a well-known single-user tool, XMind.**

I. Introduction

Collaborative applications can be designed and implemented from scratch or by using existing applications to support real-time collaboration. Implementing a groupware system from scratch has the same advantages and disadvantages of a blank sheet: the system can be implemented with all required functionalities but, at the same time, the designers are responsible of each detail related to these functionalities and of all the aspects of users' interactions.

This choice is feasible if the development from scratch is compatible also with the specific requirements coming out from the context where the system will be used. Indeed, the collaboration functionalities are not a value in themselves but they can be useful only if adequately integrated in the (ordinary) work process and environment, otherwise these collaboration functionalities may require the users an overhead to manage the collaboration beyond the standard work flow. Moreover, it should be said that most of the work activities use specific applications, answering to specific requirements, with a long history of usage and well known procedures as well as a huge quantity of data related to these applications.

These considerations motivate the studies about the possibilities of making collaborative the single-user applications, a well known approach that dates back to the 90s [1]. In fact, multi-user applications, specifically designed to support collaborative functionalities, exhibit a clear disadvantage compared to single-user applications: groupware features are used less frequently than single-user activities. As a consequence, users have no or little incentive to abandon their favorite individual application for a new application that, in addition, may show compatibility issues. The idea, therefore, is to investigate approaches that allow to efficiently add groupware features to single-user applications.

Making collaborative a single-user application implies a great saving of time and development efforts: we can reuse advanced applications (traditional office automation software, like Microsoft Word, Powerpoint, etc.) in a collaborative manner without developing them again to add collaboration functionalities. Of course, crucial in this process is the choice of the distributed architecture to be employed in making multiuser a single-user application.

In this paper, we present a distributed architecture named Centralized Control, based on the Model-View-Controller (MVC) design pattern, which shows several advantages with respect to the repertoire of available architectures that have been presented in literature. In particular, our architecture is able to add group semantics to the operations performed (enlarging, therefore, the functionalities of the application) while preserving the independency from the application itself (no modification is needed to the code) and from the underlying operating system. To wit the advantages of our architecture we show a significant example of its application, making collaborative XMind, a well-known mind-mapping tool. Then, we will compare our architecture to the state of the art of the currently known techniques to make collaborative a single-user application, in order to provide a detailed comparison and to show how the Centralized Control is, indeed, different by the previously known models.

Structure of the paper: Firstly, we introduce the mind maps and some examples of collaborative software for mind-mapping, including XMind (in Sec. II) and then, we show CollabXMind (in Sec. III), the collaborative version of XMind. The architecture of CollabXMind is then described in Sec. IV. Finally (in Sec. V), we will compare our architecture to the state of the art of the currently known techniques to make collaborative a single-user application, in order to provide a detailed comparison.

II. Collaborative mind maps

A *mind map* [2] is a graphic representation of ideas structured around a central theme. Each idea is represented graphically as a node of the map and can be linked to other nodes with a semantic relationship. For this reason mind maps highlight the semantic interconnections between ideas. Each idea can be enriched with images, hyperlinks to Web pages or other resources. In this way mind maps try to improve the "recall", the human capacity to retrieve information from the past. Mind maps are useful for the organization of ideas, especially for the activities of note-making and note-taking.

Mind maps are strongly dynamic because they are mainly used when the organization of information and ideas are not clear. Mind-mapping software systems support the instantaneous reorganization of ideas with the drag-and-drop of nodes. Software applications for mind-mapping are commonly used to create diagrams which represent ideas in relationship with other concepts or different kinds of data. A quite exhaustive list of these kind of applications can be found on Wikipedia [3].

The research in CSCW and CSCL that explores the potentialities of collaboration is using, as one of the tools, the software for collaborative mind-mapping [4], [5] that seems to exhibit several advantages, especially in brainstorming activity. Recently, also, a quantity of Web-based tools for mind-mapping have been developed, such as Mindjet Catalyst [6], MindMeister [7] and bubbl.us [8], most of which allow collaborative work. The simplest tools allow collaboration only in turn-taking style, while others support synchronous collaboration. A comparison between turn-taking access and parallel access in collaborative mind-mapping has been experimented in 2002 by Prante et al.[4], who conclude that turn-taking blocks the generation of the ideas and also leads to a structure blocking effect.

Some systems enforce the collaboration by providing other communication tools (chat or Web conferencing). In general, the limitations of Web-based applications are that they explicitly need an infrastructure (a Web server for hosting the collaboration) and furthermore, some privacy problems may arise in case of an external Web server. Moreover, an Internet connection should be available, and, in some cases, this can be unreasonable, since some organizations may limit the Internet access via specific filtering and firewall policies.

In 2009, Shih et al. [5] presented an experiment on the GroupMind system, a collaborative software for mind-mapping based on FreeMind [9]. The experiment analyzes the impact of collaborative mind-mapping on generating ideas. Two group settings have been tested: an *interaction group*, in which members can interact with one another to generate new ideas, and a *nominal group*, in which each member has a different task assigned, and then the ideas generated are combined together. The experiment has been run in two different ways: using the GroupMind system or a traditional whiteboard. The evaluation has been based on the number of unique ideas generated, which is one of the most measured performance indicators in brainstorming processes[10], [11], [12], [13]. The results show that GroupMind has performed better than the whiteboard in both group settings, so confirming the advantage of using a collaborative mind-mapping system.

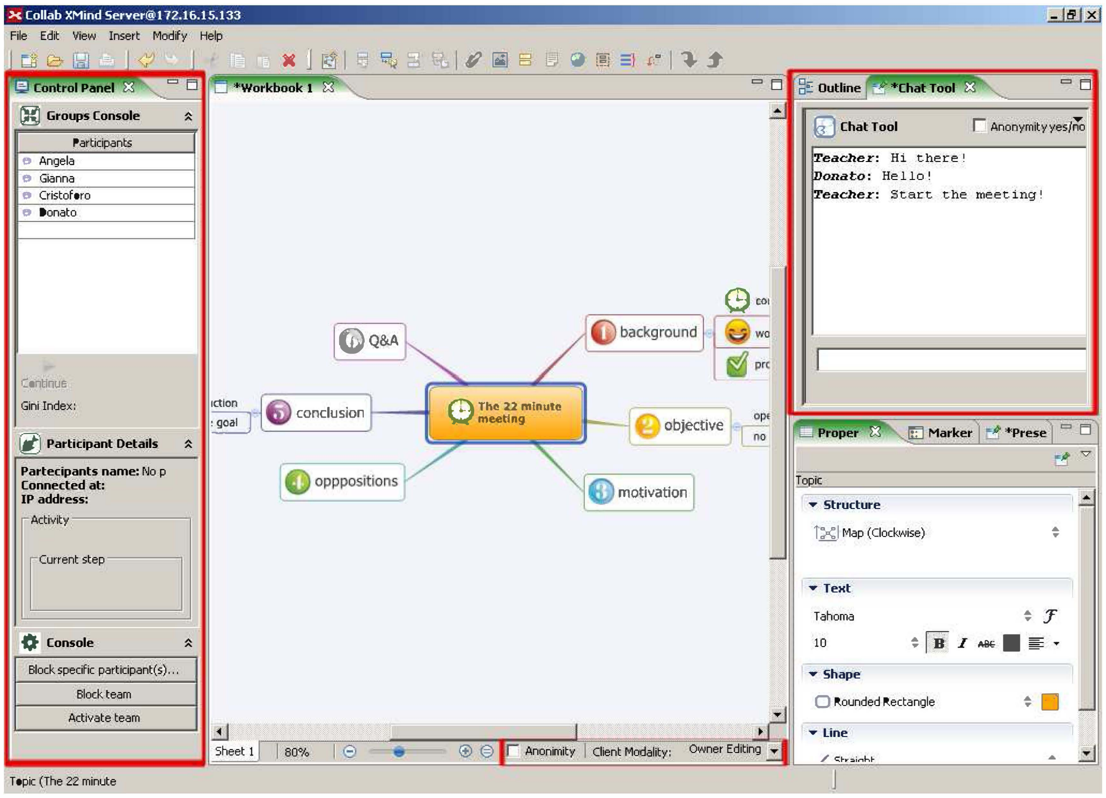
A limitation of GroupMind system, besides the fact that is still a prototype as referred in the article, and of similar systems, such as the commercial product Visual Mind [14], is that they only support an open interaction mode, in which each user can do anything. In a sense, there is no semantic difference between single-user mode and multiple-user mode, that is, in practice, only the multiplexing of the same single-user action with the same semantics.

The last tool is XMind [15], a single-user standalone mind-mapping software system that enables the user to create his own mind-maps and that we have used as an example to convert single-user applications to multi-user applications. XMind is based on the metaphor of a workbook that contains multiple sheets. The user can create his own mind-map on a sheet: around the central topic grows a graphic representing related ideas and concepts. The appearance of each item can be customized with icons, colors and so on. One of the most interesting feature is the opportunity to change the structure of interconnected ideas: the structure can represent a map, a tree, a logic chart, a fishbone and a spreadsheet. XMind is an open source project and is designed as an Eclipse-based application: it uses the core of Eclipse, the Rich Client Platform. XMind has been named the 'Best Commercial Eclipse Rich Client Platform (RCP) Application' in 2008 in the Eclipse community, and the 'Best Project for Academia' in 2009 in the SourceForge community.

III. CollabXMind

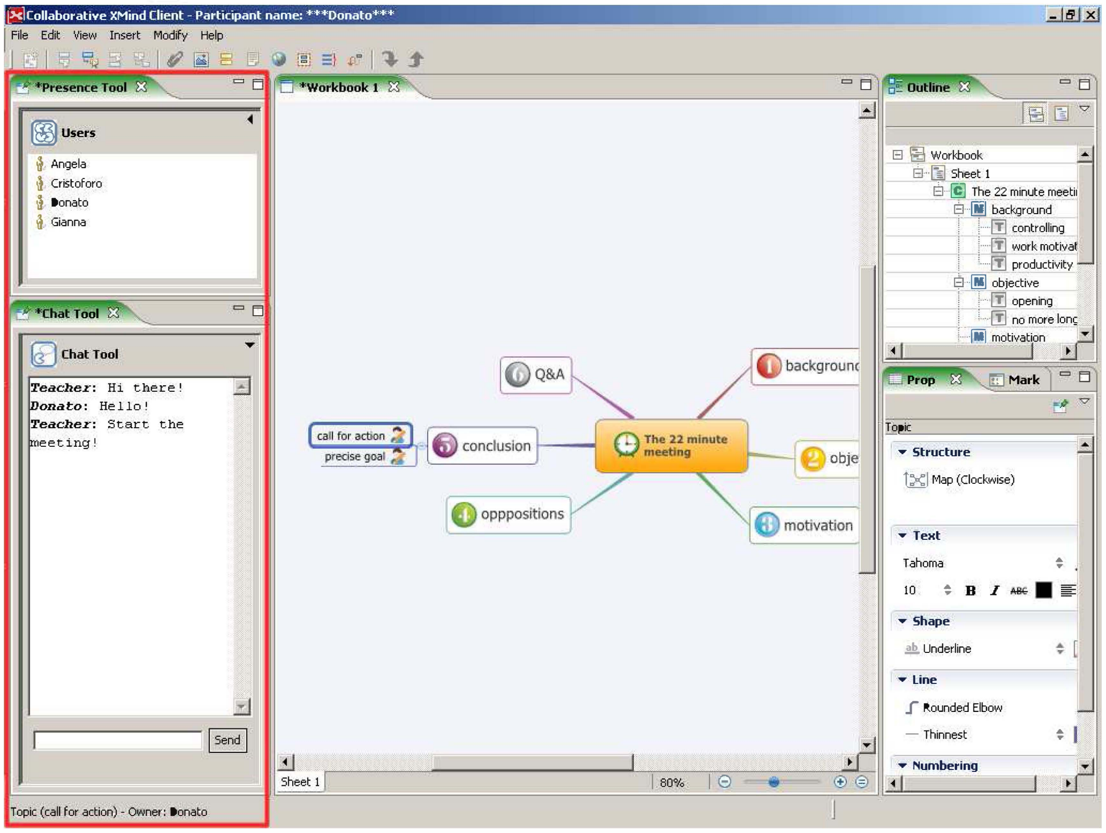
A wide community of users use XMind every day and are familiar with its features and functionalities. For this reasons, our idea is to convert the single-user application XMind into collaborative one: we aim to design and develop a collaborative real-time mind-mapping application named *CollabXMind* that enables multiple users to cooperate in parallel way on a shared map. In particular, CollabXMind enables multiple participants that are into same place at the same time, to simultaneously contribute on the same map. CollabXMind is a synchronous face-to-face collaborative system in which each member (*Participant*) can contribute adding new ideas on a shared map, created and managed by the *Coordinator*. CollabXMind is a real-time system, in which each Participant can contribute adding a new idea to map that is visible immediately to other users.

CollabXMind's user interface is similar to XMind mind-mapping application, so CollabXMind inherits most of XMind's features and preserves the original usability. Fig. 1 shows CollabXMind on the Coordinator side, where the classic menu and tool bars of XMind are shown on the top. In the middle, a collaborative workbook is shown with an example of a mind map created with contributions from all the users. Obviously, each collaborative system must have team-awareness features: in our system the Control Panel has a list of all connected Participants. The Floor control in our system



```
Teacher: Hi there!
Donato: Hello!
Teacher: Start the meeting!
```

Fig. 1. A screenshot of CollabXMind on the Coordinator side.



```
Teacher: Hi there!
Donato: Hello!
Teacher: Start the
meeting!
```
Topic (call for action) - Owner: Donato

Fig. 2. A screenshot of CollabXMind on the Participant side.

allows the blocking/unblocking of a specific user, a group of users or all Participants; all floor control operations are in the bottom part of the Control Panel. When the Coordinator blocks a Participant, Participant's user interface will be frozen. Our system provides also a chat tool that supports discussion between users; it is placed on the right of the user interface. Our system supports anonymity, since it is known to increase the number of contributions from Participants. Besides, the Coordinator at any time can change the Participants' interaction mode. Both features can be controlled by specific widgets near to the zoom control bar.

On the Participant side, CollabXMind inherits all functionalities of XMind except the operations that concern the creation/opening/import of collaborative workbooks, managed by the Coordinator. On Participants' user interfaces there are two views: the chat tool and the presence tool, respectively, for discussions support and team awareness (see Fig. 2).

A. Interaction modes

In a collaborative system it is necessary to coordinate users and define when and how they can work on the artifacts: in different moments of the collaboration, different *interaction modes* may be used, according to the goals to perform [16]. As an example, at the beginning, one may want to allow Participants to study and read the whole map, before they start to contribute. In a successive phase, for example, a free access to the map may be granted for brainstorming, where each Participant can create his/her own branch of mind map that represent his/her thoughts. To finish our example, in a final phase, the Coordinator may want to "refine" the map, allowing a face to face discussion, where the Coordinator is the one that merges, reduces and summarizes branches.

In CollabXMind, each node is associated with its creator, and the operations that can be done, are dictated by the "interaction modes". CollabXMind provides four interaction modes: *clone*, *view*, *owner editing* and *open editing*. A very useful feature of our system, is that Participants interaction mode can be changed in any time during collaborative session by the Coordinator.

In the *clone interaction mode* the whole user interface of Participants is disabled. Each Participant sees exactly the same map of the Coordinator side and all users' inputs are discarded, included zoom and bars scrolling. Obviously, the XMind workbook of Participants has exactly the same map content of the Coordinator. Indeed, each map editing performed by coordinator triggers an immediately updates of mind map on Participants maps.

With the *view interaction mode*, each Participant can now zoom and scroll independently from the Coordinator, can hide/show branch of the map, and all the visualization operations. The content is still dictated by the Coordinator, but the read access depends on each Participant's needs.

An increased collaborative mode is the *owner editing mode*, where each user can add nodes in parallel. It is a mode that aims at bringing the parallelism in contributions to Participants, while retaining a certain control over the editing of the nodes. In fact, each node is given an owner (the Participant who created it) and the modifications (editing, deletion, change of style, shape color, etc.) are allowed only to owners. This is an important characteristics of CollabXMind that we would like to emphasize: with this architecture it is possible to overimpose additional group semantics to the model, as we did in CollabXMind by adding the owner, and imposing a policy over this information. The effect is important for brainstorming: every Participant can see others' contributions, can add nodes below it, place connections, but cannot modify its content and appearance. As shown in Fig. 2, in the owner editing mode, as visual cue, all nodes owned by Participant on its host are marked with an icon representing a man with a pencil, to distinguish from other nodes that cannot be edited.

Finally, of course, in the *open editing mode* where everybody is free to modify every aspect and content of the map.

IV. CollabXMind: the architecture

XMind is an open source application founded on the core of Eclipse, a component-based Java IDE which provides its core to build plug-in based applications named *rich client applications*. XMind is implemented in Java, is multi-platform and does not provide API adequate to introduce collaboration functionalities. Our aim was to introduce collaboration functionalities allowing multiple users to work concurrently on the same mind map without modifying the original application, so that an easier update is possible with new versions of XMind.

Our approach introduces collaboration functionalities by intercepting the users'input in the Model-View-Controller implementation of XMind and then the input intercepted is passed to the framework providing communication functionalities. Our solution does not change the source code of the original application and use a framework which provides support to achieve advanced collaboration functionalities. In the next section we provide the detailed description of the architecture of CollabXMind, whose overview is shown in Fig. 3).

The architecture is partially replicated: each user has an instances of CollabXMind (the Client in Fig.3) and the users'input are merged in a single stream managed by the server of CollabXMind. Indeed, the server is responsible of the merging of users'input and of applying policies.

XMind implements the Model-View-Controller pattern to separate the data visualization from the data model. The other layers are the components we have developed to make collaborative XMind without any change on the original application or its source code. The basic idea is to intercept the events of the application and pass them to a framework which provides the collaboration infrastructure:

- the CollabXMind layer is responsible of intercepting the users'input and to pass them to the communication layer and vice versa; on the server side, it applies policies and access control;
- the CAFE tool server (/client) is the layer responsible of connecting the CollabXMind layer with the communication layer;

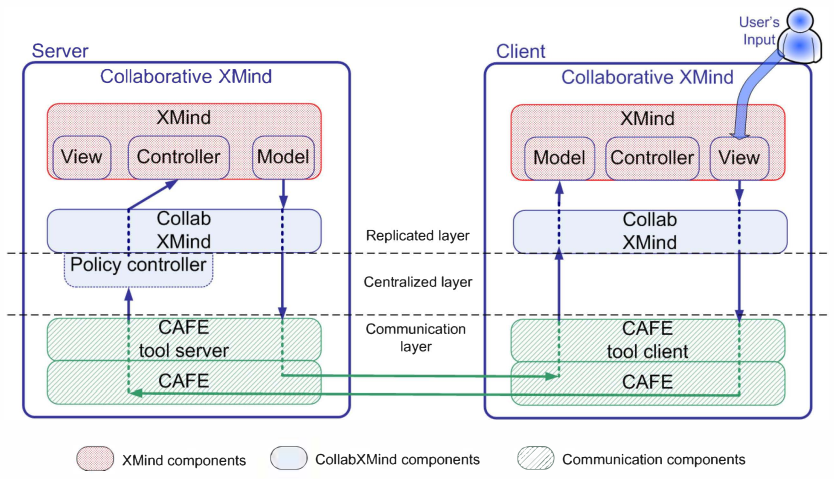


Fig. 3. The Architecture of CollabXMind is composed by the original application XMind, and by a layer CollabXMind responsible of bridging the original application with the communication layer.

- the CAFE layer is responsible of communication functionalities and of advanced collaboration features.

In this architecture, we identify a server application and a client application, with a partially replicated design: the XMind application and the CollabXMind layer are replicated on each instance of CollabXMind, while the Policy controller is centralized on the server, and is responsible to manage the sequence of events (to ensure the state synchronization among the replicas) and to apply policies and access control. The design of CollabXMind layer and the functionalities provided by CAFE are described in detailed in the following subsections. Further functionalities can be envisioned both in CAFE and in CollabXMind according to the aim to achieve.

A. The CollabXMind layer

The CollabXMind layer is responsible of intercept the events of the XMind application. An high level description of the mechanism is depicted in Fig. 3, which describes how the input provided by user playing as client is shared. The user provides input to its local View component; the user input generates a *request* that is caught by CollabXMind layer and passed to the communication layer which sends the request to the server; the server side of the CollabXMind enqueues the request (to ensure synchronization among all the requests), then it checks if the user is allowed to execute the request and eventually it passes the request to the controller of XMind; the Controller of XMind processes the request as it comes from the local View and updates the Model, which notifies all its listeners of the change; the CollabXMind layer has a listener on the Model which serializes the notified event and passes it to the communication layer; the communication layer sends the event to all the clients; the CollabXMind layer of the clients handles the event updating the Model of XMind, which updates the View (through the Controller).

XMind implements the pattern Model-View-Controller through GEF (Graphical Editing Framework) [17]. GEF is an

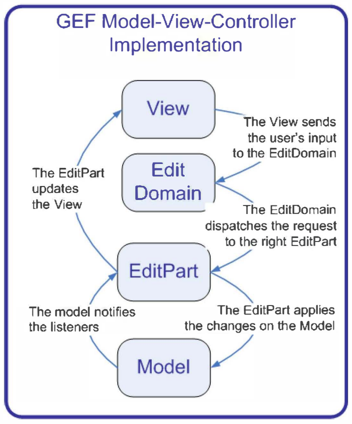

GEF Model-View-Controller Implementation
View
The View sends the user's input to the EditDomain
Edit Domain
The EditPart updates the View
The EditDomain dispatches the request to the right EditPart
EditPart
The model notifies the listeners
The EditPart applies the changes on the Model
Model

Fig. 4. The GEF implementation of the Model-View-Controller pattern.

Eclipse plugin[1] supporting the creation of graphical editors. The developer can then take advantage of the many common operations provided in GEF and/or extend them for the specific domain. The components defined by GEF are the Model, the View and the EditPart (i.e. the Controller) and their interactions are depicted in Fig. 4. In an high level description of the system, the Model manages a set of listeners that are notified when the changes happen. The EditPart is a listener of the Model and when receives the notification events updates the View. When the View receives the user's input, it sends a request to an EditDomain component, which, on the basis of the nature of the request, dispatches the request to the right EditPart, which applies the change on the Model.

The general idea of our approach is to intervene in the implementation of the Model-View-Controller pattern adding our components to intercept the users'requests and send them to the server, which processes them and sends the resulting events to all the clients. The Fig. 5 depicts the details of the architecture built on the GEF Model-View-Controller im-

[1]Indeed, GEF is implemented as two plugins

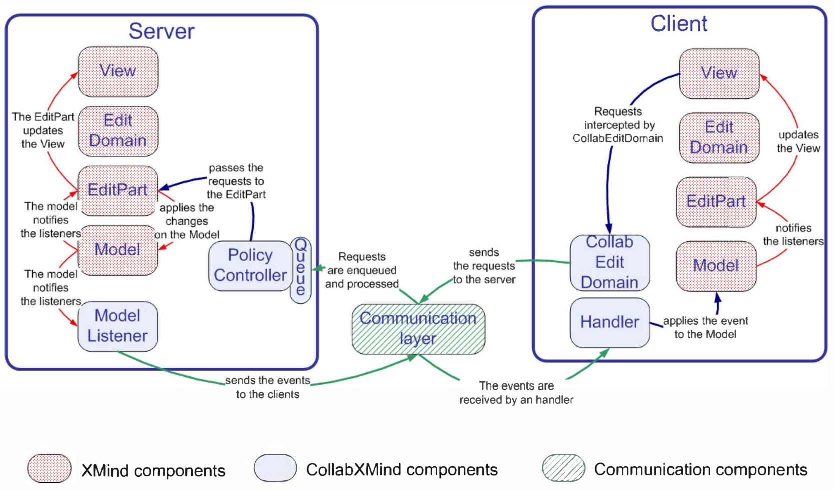

Server
View
The EditPart updates the View
Edit Domain
EditPart
passes the requests to the EditPart
The model notifies the listeners
applies the changes on the Model
Model
Policy Controller
Queue
The model notifies the listeners
Model Listener
Requests are enqueued and processed
sends the requests to the server
Communication layer
sends the events to the clients
The events are received by an handler
Client
View
Requests intercepted by CollabEditDomain
Edit Domain
updates the View
EditPart
notifies the listeners
Collab Edit Domain
Model
Handler
applies the event to the Model
XMind components
CollabXMind components
Communication components

Fig. 5. The CollabXMinds components intervene on the implementation of the model-view-controller pattern to bridge users'input and communcation layer.

plementation, illustrating the process of input of a client. On the client side, CollabXMind replaces the original EditDomain of XMind with a CollabEditDomain. This replacement does not requires changes on the source code of XMind because can be done through the API of GEF. The CollabEditDomain intercepts the requests going from the View to the EditPart (and prevents the requests to be processed by the EditPart) and sends them to EditPart on the server side through the communication layer. On the server side, all the clients' requests are enqueued and then processed by a Policy Controller added by CollabXMind. The Policy Controller checks the user's permissions and passes the request to the XMind EditPart, which applies the operations on the Model which notifies the listeners. CollabXMind has a listener on the Model which sends all the change events to the clients. On the client side, an handler defined by CollabXMind receives the change events and updates the Model. The Model notifies of the change its listeners, including the EditPart which updates the View.

The process of a request of the user playing as coordinator is slightly different: its input is normally processed by the XMind EditDomain and then by the XMind EditPart; when the EditPart updates the Model, it notifies the listeners, including the CollabXMind Model Listener which sends the event to all the clients.

B. *The Collaboration Framework*

So far we have described our solution to introduce collaboration speaking in general term of a *communication layer* which provides communication functionalities and support to achieve advanced collaboration functionalities. The framework we have used is CAFE (Collaborative Application Framework). It derives from CoFFEE [18], [19], [20], a collaborative environment to support the face to face learning. CoFFEE has a server and a client application named CoFFEE Controller and CoFFEE Discusser, which are used by the teacher and the students. CoFFEE provides a set of collaborative tools integrated in the environment to support specific aims and contexts: the Threaded Discussion tool supports tree-structured discussions, the Graphical Discussion tool provides a shared graphic workspace, the Positionometer supports votings; a detailed description of CoFFEE and its tools is provided in [20]. CoFFEE is available since July 2008 on SourceForge [19].

CoFFEE has a component based architecture, founded on Rich Client Platform (RCP), the core of Eclipse. The core of CoFFEE provides the communication and collaboration functionalities used by the CoFFEE applications and tools. The plug-in based architecture of CoFFEE inherited by Eclipse allowed us to define the environment so that each collaborative tool is implemented as a plug-in. A detailed description of the architecture of CoFFEE is provided in [18], [20]. To create CollabXMind as a CAFE based application, we have defined a CAFE tool which bridges the CollabXMind layer and the CAFE framework. The CAFE tool has a server side and a client side, and each side is integrated on CAFE through the Eclipse extension-point mechanism. In the following we describe the set of functionalities that CAFE provides to CollabXMind.

Communication functionality. Obviously, the first essential functionality is the communication. The communication is implemented in CAFE through ECF (Eclipse Communication Framework) [21]. This high level of abstraction allowed us to implement communication easily.

Server Discovery. CAFE implements an automatic mechanism to allow the clients discovering the server on the local network. This functionality simplifies the initial phase of connection of clients to the server.

Authentication. CAFE allows the initial registration of users, allowing to choose if the connection can be free or should be authenticated against a list of known users. Currently, the authentication does not implement any sophisticated security mechanism, but just a check on a list of names.

Tools life cycle management. CAFE provides the possibility to load at runtime any CAFE tool: this allows to use any CoFFEE tool within CollabXMind. Tools like the Presence tool or the Chat as well as the Threaded Discussion tool can support the meta-communication around the collaborative activities on the mind map: they can provides awareness about the team composition (presence tool), a channel for simple communications (i.e., the chat), a textual structured space (the Threaded Discussion tool).

Team Awareness. CollabXMind uses on the server side a view named Control Panel provided by CAFE, showing the Participants list. The CollabXMind client currently can have feedback about the list of Participants through the Presence tool provided by CoFFEE.

Floor Control. The floor control provided by CAFE allows the coordinator to block/unblock selectively the users: by the Control Panel the coordinator can block/unblock a single user, a group of users or all the team.

Latecomers management. The latecomers management is provided by CAFE to support the late joining of clients and the synchronization of their state. Details about the mechanism implemented are presented in [18].

V. Distributed Architecture for making collaborative single-user applications

Many studies have faced the question of introducing collaboration functionalities in existing single-user applications. A review and a comparison of existing approaches to make a single user collaborative is provided in [22]: they organize the existing studies the basis of the approach used to introduce collaboration in single-users systems.

An approach involves the need to modify the source code of the existing single-user application. Obviously, this requires the availability of the source code and the appropriate license. The evident drawback is represented by the need to update the multi-user application source code for any update of the single-user counterpart.

Toolkits (TO): In literature several studies describe the use of toolkits to support the creation of new collaborative application. An examples of work based on this approach is DistEdit [23], a groupware toolkit to convert existing single-user editors (MicroEmacs and GNU Emacs) into group editors by requiring little changes on the source code of the original editors. In another work [24] it is also addressed the problem of interoperating groupware systems by providing concurrency control policies. It proposed to build a bridge programmed to provide communication and interoperation functionalities. However, it assumes the internal source code and groupware applications knowledge. Several works on groupware have employed the MVC to realize collaborative architectures. In particular, the Clock groupware toolkit implements a semi-replicated MVC architecture as described in [25]. It support the development of synchronous groupware in high-level MVC style: the model is implemented on a centralized server while each user's view/controller is implemented on a separate client node. Another example of using MVC to realize collaborative architecture is implemented in the Rendevous framework [26], that provides a centralized MVC architecture and a programming language to simplify the construction of applications for real-time collaboration. The approach proposed in this work is characterized by a model centralized on a server node and shared by several remote client views.

Shared Windows Systems (SWS): these systems, also known as Collaboration Transparency Systems, were one of the first approaches to make collaborative single-user applications [1] without access and modify the source code. Examples include SharedX [27], Microsoft Meeting Space [28] and SunForum [29]. The aim was to develop a generic solution to make collaboration functionalities available to any single-user application. The idea is to intercept at operating system level the users input and share it by means of *conference agent* interposed between the application and the window system provided by the operating system. The conference agent merges the inputs coming from all the windows systems of the users, creating a single stream and sending it to the application. This kind of approach is independent from the specific application involved in the collaboration but it strongly depends on the underlying operating system (which manages the windows system). The dependency from the operating system may represent a problem if the application involved in the collaboration is multi-platform.

Intelligent Collaboration Transparency (ICT): The Intelligent Collaboration Transparency [30], an evolution of the approach described before, has been specifically designed for heterogeneous applications of the same category (such as text editors). Both the the ICT and SWS approaches are independent from the specific application involved in the collaboration but are strongly dependent from the underlying operating system. This dependence may represent a problem if the application involved in the collaboration is multi-platform.

Component Replacement (CR): The Component Replacement (also named Flexible Collaboration Transparency) [31] approach shifts the focus from the operating system level to the application level, defining a replicated architecture where each user has a replica of the application. The collaboration is introduced without modify the source code of the original application through the replacement of selected single user components ('the combination of data and behavior') of the application with collaborative ones. The goals of this approach were to introduce collaboration allowing unanticipated sharing, support concurrent work and relaxed WYSIWIS, providing group and work awareness, accommodating late comers. The runtime component replacement proposed in this approach requires that the original application uses a platform allowing the runtime replacement of components and the dynamic binding (the runtime resolution of a function invocation or data access); the accommodation of late comers through the image copy of the shared application requires the support of the process migration; the synchronization among the replica happens through the merge of users'input in a single stream (at each replica). This approach has been realized in Flexible JAMM [31], implemented on Java and based on JOS (Java Object Serialization) to support the process migration and on Swing to support the component replacement. As the authors recognize in the limitations of the approach, several constraint coming from the method (the runtime component replacement) and from the implementation (based on Java, Swing and JOS) restrict the set of shareable applications to a subset of serializable Swing based Java application.

***Transparent Adaptation (TA)**:* This approach is oriented to specific applications and is not generic as shared windows systems. The idea is to introduce collaborative functionalities by using its API without modifying the original application. This approach has been used for example in CoWord and CoPowerPoint [32], [33] to allow multiple users to view and edit any objects at the same time over the Internet. In the transparent adaptation approach the developers do not need to change the source code of the application. This approach is based on the use of the application APIs to get users's local interactions and on Operational Transformation technique to ensure consistence among the replicas; the approach requires that the single user application provides API suitable to intercept the input events and that the API and the application data are adaptable to the operational transformation which ensure the synchronization and consistency among the replicas.

***Flexible Coupling (FC)**:* This approach faces a new aspect about the introduction of collaboration features in single user applications: the users participating in the collaboration may wish to share selected parts of the application content. This approach has been applied as case study to introduce collaboration in GraphDraw [34]. In the case study, the users may wish to share the graph that they are creating but not their personal annotations. The basic idea of the Flexible Coupling approach is to define several layers corresponding to several aspects of the collaboration: for instance, in the case study of GraphDraw the layers were the graph (corresponding to the model in the MVC architecture), the graph view (corresponding to the manipulable graphic representation of the graph), the figures layer (corresponding the users' annotations), the appareance (corresponding to the graphic widgets which do not affect the model, like the scroll bar) and the window (corresponding to all the previous together). This approach allows the users to choose which layer should be shared: for instance they can share the graph view but not the figures.

The Flexible Coupling requires to define specific layers for each application in which the programmers are introducing collaboration and requires changes to the source code of the original application.

***Component Mapping (CM)**:* This approach discusses mappings to extend single-user application to support collaborative activities. This approach has been used to convert ArgoUML, a widely used open-source CASE (Computer Aided Software Engineering) tool to a multi-user tool called CoArgoUML [35]. This mapping assumes that the single-user application has been implemented using the MVC architecture style and that changes on the source code of the original application will be required. This paper derives that four minimal collaborative requirements should be applied to single-user applications to make them collaborative, that is, communication, group awareness, session management and concurrency control.

These approaches have been studied and classified and their classification [22] is presented in table I. The table reports for each approach the requirements on the *Source Code* (no need of source code, need of OS API, need of application API, need of knowledge of the source code, need modifications on the source code), about technical requirements (need to use a specific language, need to develop a software layer, need of a specific architecture) and the final MVC architecture. The last column, MVC architecture, indicates the model resulting in the MVC pattern for each approach, according to Pichiliani [22] on the basis of the MVC architectures presented by Suthers in [36]. The defined architecture are Centralized, Replicated, Distributed and, finally, Hybrid and are shown in Fig. 6, models a), b), c) and d).

The table reports for each approach the requirements on the *Source Code* (no need of source code, need of OS API, need of application API), about technical requirements (need to use a specific language, need to develop a software layer, need of a specific architecture) and the final MVC architecture. The last column, MVC architecture, indicates the model resulting in the MVC pattern for each approach, following the MVC architectures presented by Suthers in [36]. The defined architecture, shown in Fig. 6 are Centralized, Replicated, Distributed and, finally, Hybrid.

In all the cases, both Suthers [36] and Pichiliani [22], describe the distribute model of the MVC pattern as a result of the introduction of collaboration in existing systems.

A. Centralized Control Architecture

Our approach, reported in the last row of table I, tries to introduce collaboration in a single user application by intervening in the implementation of the MVC pattern to intercept the users'input defining a *Centralized Controller* architecture (*CC*), shown in Fig.6, model (e). Our architecture does not need to modify the source code, even if we need the source code to understand which are the components that implements the MVC pattern. Our approach has no need of APIs neither of the OS nor of the application, so it is operating system independent and is suitable for applications which does not provides adequate API. The requirement to access to the source code and the no need of API make the approach suitable for open source applications (as XMind) but not for commercial application (as Word).

The implementation of our approach has been presented in detail in the previous section, and here we want just to describe and summarize its principles: the Centralized Control MVC architecture. As shown in Fig. 6, each user has a replica of the View, Model and of the Control display (the Controller part responsible of updating the View on the changes of the Model). Users' input on the replicated Views are sent to the Centralized Control. The Centralized Control applies the changes on the local View and Model, and the latter sends the changes to the replicated Models. Finally, the replicated Models update, by means of the Control display, the local Views. The cornerstone of the architecture is the Centralized Control: it is responsible of the events ordering and then of the consistence among

Approach	Source code	Technical Reqs.	MVC Arch.
Collaboration Transparency Systems	No	specific lang. prog.	Centr
Intelligent Collaboration Trasparency Systems	OS API	need software layer	C or D
Trasparent Adaptation	App.API	new software layer	R
Component Replacement	knowledge of source code	components progr language	R
Flexible Coupling	modifications on source code	definition of layers	R
Component Mapping	modifications on source code	MVC pattern	H
Centralized Control Architecture	knowledge of source code	MVC pattern	CC

TABLE I
COMPARISON OF THE TECHNICAL ASPECTS OF THE APPROACHES TO MAKE COLLABORATIVE SINGLE-USER APPLICATIONS.

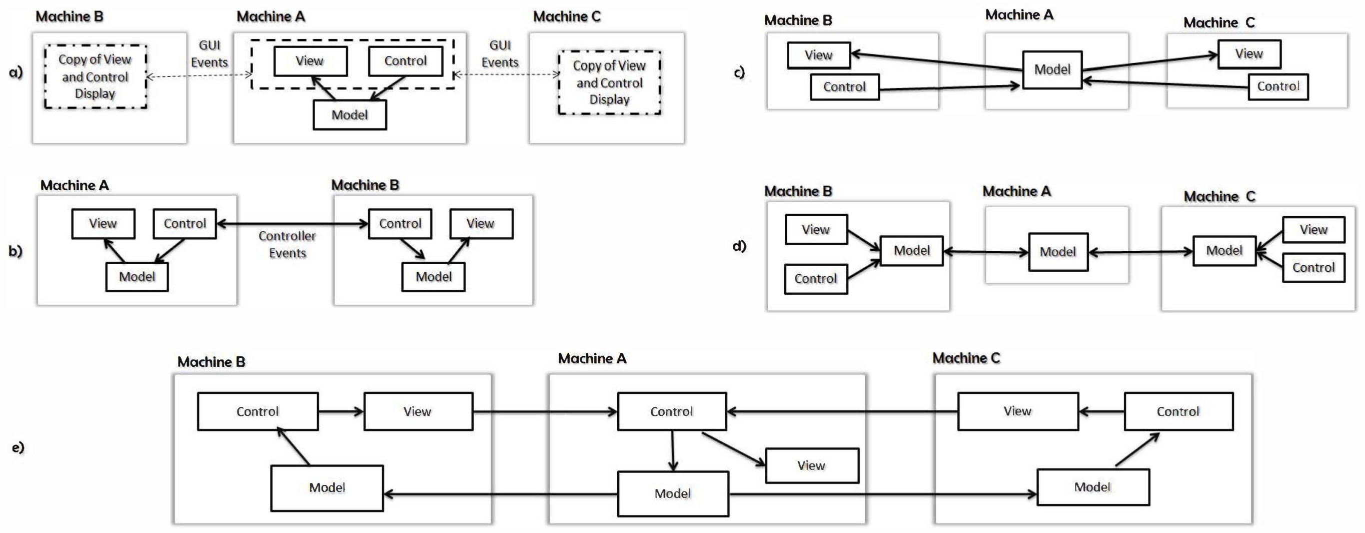


Fig. 6. MVC-style architectures: a) Centralized architecture [36], b) Replicated architecture [36], c) Distributed architecture [36], d) Hybrid architecture [36], e) Centralized Control architecture (this paper).

the replicas, it has the possibility to applies policies and to add semantic to the application events (like the author of the command, in our example for CollabXMind).

In the comparison with other models, our approach presents some advantages and some drawbacks. Our approach provides concurrent work and relaxed WYSIWIS, and allows us to apply semantic to the operations (like the author of the operation) and then (editing) policies. Our approach does not require modifications on the source code, has no need of OS API (so it is independent from the OS and can be applied to multiplatform applications) and has no need of the application API, so it is suitable to application which does not provide API. On the other hand, our approach requires that the original application implements the MVC pattern and that provides the source code, even if we do not need to modify the original application, but we need to understand how tht pattern is implemented to intercept the users'events.

VI. Conclusions

In the CSCW and CSCL fields several approaches and examples have been proposed to introduce collaboration in single-users applications, with different advantages and drawbacks. The collaborative architectures resulting from the techniques presented in literature use the MVC architectures defined by Suthers in [36]. Our approach defines a new MVC architecture, named *Centralized Control*, with the aim of introducing collaboration in single-users applications without modify the source code of the original ones, by intervening in the implementation of the MVC pattern to create a collaborative structure with a centralized controller. Compared with other techniques, the Centralized Control technique allows us to add semantic information to the users'operation and to apply editing policies on the basis of these information. Our technique has been implemented to introduce collaboration functionalities in XMind, an Eclipse-based single-user application to create mind maps. XMind is open source and does not provide API suitable to introduce collaboration. It implements the MVC pattern via GEF. The implementation of the Centralized Control technique has allowed us to introduce collaboration without modify the source code and to provide a flexible set of editing policy. The main drawbacks of the Centralized Control technique are the requirement of the implementation of the MVC pattern on the original application and the availability of the source code to understand how the pattern is implemented. On the other hand, this technique is OS independent and does not requires application API, so it seems more suitable for open source applications (as XMind) rather than for commercial applications (as Microsoft Word or Power Point). The implementation of the Centralized Control technique has produced CollabXMind, an application that allows multiple users to collaboratively work on a shared mind map. CollabXMind has been developed using CAFE as underlying layer that provides collaboration functionalities. Here we want also emphasize that, beyond the communication and collaboration functionalities, CAFE has enriched the final result by providing the possibility of using other CAFE-based tools (coming from the CoFFEE environment), providing further collaboration channels.

Further studies will concern both CollabXMind and the Centralized Control approach: of course, future works will include experimentations to evaluate the effectiveness and the usability of the integration of the collaboration features in CollabXMind. Moreover, further studies about the Centralized Control approach will be on the possibility of developing a generic implementation able to introduce the collaboration in the MVC pattern. First steps in this direction will be about the GEF-based implementation, to develop a *collaborative* GEF by generalizing our current implementation to offer collaborative features for other GEF-based applications.

References

[1] J. C. Lauwers, T. A. Joseph, K. A. Lantz, and A. L. Romanow, "Replicated architectures for shared window systems: a critique," *SIGOIS Bull.*, vol. 11, no. 2-3, pp. 249–260, 1990.

[2] T. Buzan and B. Buzan, *The Mind Map Book: Available from Amazon.comShared Visions Unlimited Reviews Home PageHow to Use Radiant Thinking to Maximize Your Brain's Untapped Potential.* E. P. Dutton, 1994.

[3] "Wikepedia." [Online]. Available: http://en.wikipedia.org/wiki/List_of_mind_mapping_software

[4] T. Prante, C. Magerkurth, and N. Streitz, "Developing cscw tools for idea finding -: empirical results and implications for design," in *CSCW '02: Proceedings of the 2002 ACM conference on Computer supported cooperative work.* New York, NY, USA: ACM, 2002, pp. 106–115.

[5] P. C. Shih, D. H. Nguyen, S. H. Hirano, D. F. Redmiles, and G. R. Hayes, "Groupmind: supporting idea generation through a collaborative mind-mapping tool," in *GROUP '09: Proceedings of the ACM 2009 international conference on Supporting group work.* New York, NY, USA: ACM, 2009, pp. 139–148.

[6] Mindjet Catalyst. [Online]. Available: http://www.mindjet.com/products/mindjet-catalyst/overview

[7] MindMeister. [Online]. Available: http://www.mindmeister.com

[8] bubbl.us. [Online]. Available: http://www.bubbl.us

[9] FreeMind. [Online]. Available: http://freemind.sourceforge.net

[10] T. Bouchard, "Personality, problem-solving procedure, and performance in small groups." *Journal of Applied Psychology*, vol. 53, 1, pp. 1–29, 1969.

[11] R. Gallupe, A. Dennis, W. Cooper, J. Valacich, L. Bastianutti, and J. Nunamaker, "Electronic brainstorming and group size." *The Academy of Management Journal*, vol. 35, 2, pp. 350–369, 1992.

[12] M. Diehl and W. Stroebe, "Productivity loss in brainstorming groups: Toward the solution of a riddle." *Journal of Personality and Social Psychology*, vol. 53, 3, pp. 497–509, 1987.

[13] R. O. Briggs, B. A. Reinig, and M. M. Shepherd, "Quality as a function of quantity in electronic brainstorming," in *HICSS '97: Proceedings of the 30th Hawaii International Conference on System Sciences.* Washington, DC, USA: IEEE Computer Society, 1997, p. 94.

[14] Visual Mind, Mind Technologies. [Online]. Available: http://www.visual-mind.com/

[15] "Xmind." [Online]. Available: http://www.xmind.net/

[16] S. Noël and J.-M. Robert, "Empirical study on collaborative writing: What do co-authors do, use, and like?" *Computer Supported Cooperative Work (CSCW)*, vol. 13, pp. 63–89, 2004, 10.1023/B:COSU.0000014876.96003.be. [Online]. Available: http://dx.doi.org/10.1023/B:COSU.0000014876.96003.be

[17] "Graphical Editor Framework," last access on june 2010. [Online]. Available: http://www.eclipse.org/gef/

[18] R. De Chiara, A. Di Matteo, Ilaria Manno, and V. Scarano, "CoFFEE: Cooperative Face2Face Educational Environment," in *Proceedings of the 3rd International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2007), November 12-15, 2007, New York, USA*, 2007.

[19] CoFFEE, "CoFFEE at Sourceforge:," http://sourceforge.net/projects/coffee-soft, 2010. [Online]. Available: https://sourceforge.net/projects/coffee-soft/

[20] R. De Chiara, I. Manno, and V. Scarano, *In Educational Technologies for Teaching Argumentation Skills.* Bentham eBooks, in press, ch. CoFFEE: an Expandable and Rich Platform for Computer-Mediated, Face-to-Face Argumentation in Classroom.

[21] ECF, "Eclipse Communication Framework," 2010, http://www.eclipse.org/ecf/. [Online]. Available: http://www.eclipse.org/ecf/

[22] M. C. Pichiliani and C. M. Hirata, "A technical comparison of the existing approaches to support collaboration in non-collaborative applications," *Collaborative Technologies and Systems, International Symposium on*, pp. 314–321, 2009.

[23] M. J. Knister and A. Prakash, "Distedit: a distributed toolkit for supporting multiple group editors," in *CSCW '90: Proceedings of the 1990 ACM conference on Computer-supported cooperative work.* New York, NY, USA: ACM, 1990, pp. 343–355.

[24] P. Dewan and A. Sharma, "An experiment in interoperating heterogeneous collaborative systems," in *Proceedings of the Sixth European conference on Computer supported cooperative work.* Norwell, MA, USA: Kluwer Academic Publishers, 1999, pp. 371–390.

[25] T. C. N. Graham, T. Urnes, and R. Nejabi, "Efficient distributed implementation of semi-replicated synchronous groupware," in *UIST '96: Proceedings of the 9th annual ACM symposium on User interface software and technology.* New York, NY, USA: ACM, 1996, pp. 1–10.

[26] R. D. Hill, T. Brinck, S. L. Rohall, J. F. Patterson, and W. Wilner, "The rendezvous architecture and language for constructing multiuser applications," *ACM Trans. Comput.-Hum. Interact.*, vol. 1, no. 2, pp. 81–125, 1994.

[27] D. Garfinkel, B. Welti, and T. Yip, "HP Shared X: a tool for real time colalboration," *HP Journal*, vol. 45,2, pp. 23–36, 1994.

[28] "Windows Meeting Space." [Online]. Available: http://www.microsoft.com/india/windows/windows-vista/features/meeting-space.aspx

[29] "SunForum." [Online]. Available: http://docs.sun.com/app/docs/doc/805-4479-12

[30] 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.

[31] 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.

[32] S. Xia, D. Sun, C. Sun, D. Chen, and H. Shen, "Leveraging single-user applications for multi-user collaboration: the coword approach," in *CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work.* New York, NY, USA: ACM, 2004, pp. 162–171.

[33] C. Sun, S. Xia, D. Sun, D. Chen, H. Shen, and W. Cai, "Transparent adaptation of single-user applications for multi-user real-time collaboration," *ACM Trans. Comput.-Hum. Interact.*, vol. 13, no. 4, pp. 531–582, 2006.

[34] V. Roussev and P. Dewan, "Supporting high coupling and user-interface flexibility," in *ECSCW'05: Proceedings of the ninth conference on European Conference on Computer Supported Cooperative Work.* New York, NY, USA: Springer-Verlag New York, Inc., 2005, pp. 45–64.

[35] M. Pichiliani and C. Hirata, "A guide to map application components to support multi-user real-time collaboration," *International Conference on Collaborative Computing: Networking, Applications and Worksharing*, vol. 0, p. 2, 2006.

[36] D. Suthers, "Architectures for computer supported collaborative learning," in *Proc. of the IEEE International Conf. on Advanced Learning Technologies (ICALT 2001)*, 2001, pp. 25–28.