A GUIDE TO MAP APPLICATION COMPONENTS TO SUPPORT MULTI-USER REAL-TIME COLLABORATION

Mauro C. Pichiliani and Celso M. Hirata
Department of Computer Science
Instituto Tecnológico de Aeronáutica
São José dos Campos, Brazil, 12.228-90
{pichilia,hirata}@ita.br

Abstract - **Building a collaborative application from scratch is a hard task. In the last decade many advances have been made to help the developers to construct collaborative applications, however little effort has been made to extend existing single-user applications to support real-time collaboration. This work presents a mapping from the main components of an existing single-user Model-View-Controller based application to multi-user real-time components of the collaborative application. The mapping allows reuse of existing single-user components by facilitating the construction of collaborative applications. This paper describes the mapping, the extension of an existing single-user application and discusses an experiment of the prototype application where the mapping was applied.**

I. INTRODUCTION

Building a groupware application from scratch is a hard and lengthy task, mostly because the designers of such applications need to consider the functionalities of the application and the collaborative aspects of the user's interactions. Another approach is to reuse an existing application, which the users already know and are used to it, to support real-time collaboration.

According to Xia et al. [13], leveraging single-user applications to allow multi-user collaboration has the potential to both significantly increase the availability and improve the usability of collaborative applications. Some research efforts have been made in the Computer Supported Cooperative Work (CSCW) area in the last decade to reduce the effort needed to extend and adapt existing single-user application to support collaboration. These approaches are suitable to adapt existing applications without changing their inner components, but few of them focus on a more general modification targeting the architectural style of the existing single-user application.

This paper presents a mapping to extend single-user applications to support multi-user real-time collaboration over the Internet. For this mapping, it is assumed that the single-user application is built using the Model-View-Controller (MVC) architecture style.

Using this mapping, existing single-user applications that are based on the MVC architecture style can be benefited and become collaborative. Since there is a great variety of single-user applications available based on the MVC architecture style, the mapping can help developers to implement new collaborative features on their existing applications, expanding the use of groupware to new classes of applications.

The proposed approach has been applied to convert ArgoUML, a widely used open-source CASE (Computer Aided Software Engineering) tool [1], to a multi-user real-time collaborative CASE tool, named CoArgoUML.

The paper is structured as follows. Section 2 describes the existing approaches to leverage single-user applications to multi-user collaboration. Section 3 presents the requirements and the proposed mapping to support collaboration on applications based on the MVC architectural style. Section 4 shows how the mapping is used to extend the ArgoUML tool as a proof of concept. Section 5 describes the experiment conducted to evaluate the use of the extended tool. Finally, conclusions and future work are presented.

II. RELATED WORK

Groupware toolkits make easier the implementation of new collaborative applications by providing group components and awareness-enhanced widgets (e.g. shared user interface components) designed to be reusable, allowing the creation of powerful new collaborative applications in a timely manner [3].

The reuse of the components of a groupware toolkit can help the development of a new application but these components cannot help the adaptation of an existing single-user application to become collaboration-aware. This happens because, in general, to reuse components of the toolkit the application developers need to implement very specific details of the toolkit, which is not always compatible with the most existing single-user applications. Although the toolkits cannot be used to extend existing application on most cases, they can be useful to help the application developers that desire to adapt some existing application. They can provide insights and ideas needed to make the necessary adaptations on the applications to support essential aspects of communication, collaboration and cooperation. The literature presents many well-established groupware toolkits such as GroupKit [8], COAST [11], and MAUI [7].

Collaboration-transparent systems are a general approach to provide computer support of real-time collaboration, usually synchronous collaboration in existing single-user applications. The sharing provided by this approach is unknown or "transparent" to the single-user application and its developers [2] provided by some external application that masks the necessary communications and notifications needed to support basic aspects of collaboration among multiple users at different sites. Examples of application-sharing based on the collaboration-transparency strategy are the Microsoft Netmeeting [10], SunForum 3D [12] and SharedX [5].

Intelligent Collaboration Transparency, or ICT [9], presents a variation of the collaboration-transparency system, using an application sharing infrastructure interposed between the heterogeneous applications to be shared and their window environment at each site. Due to the heterogeneity, users can collaborate on the common task with their favorite single-user applications but a specific platform event capture/replay module has to be used.

The approach of replacing some components of a single-user application by collaborative ones was first introduced by flexible JAMM (Java Applets Made Multi-user) [2]. The approach provides support of multiple styles of collaboration by dynamically replacing certain single-user interface objects with collaborative user interface components. To use these collaborative components the single-user application that receives the collaborative components must meet certain conditions, which include capabilities for process migration, run-time component replacement, dynamic binding, and user input event interception and replay [2]. The main limitation of this approach is that the interface components offered by Flexible JAMM only support the awareness aspects of collaboration, leaving to the collaborative application developer all the remaining work to adapt the single-user application.

The Transparent Adaptation (TA) was proposed to help modifying commercial applications that do not provide source code. This approach is based on the use of the shared application and its execution environment's API (Application Programming Interface) to intercept the user's local interactions, convert these interactions into abstract operations, manipulate these operations by collaborative techniques, and interpret the modified operations by means of the application's API at remote collaborating sites [13].

The term transparent is used because this approach does not require any change of the single-user application's source code, but it requires the application's API to be capable of handling the application’s data and the collaboration mechanisms.

III. Requirements and the Mapping of Components

Applications that support essential CSCW features have specific requirements that are not addressed by single user applications. CSCW researchers studied several kinds of requirements for CSCW applications over the last 20 years, from user interface requirements to architectural styles requirements. The minimal requirements described here originate from the areas of CSCW and object-oriented programming.

For the sake of simplicity only the minimal requirements that could be implemented on applications to make it somehow collaborative are addressed in this article.

Support for communication is often a requirement for synchronous CSCW systems. For distant and synchronous collaboration, in particular, it is very important for users of the system to know what other users want to do and their intentions provided by the effective communication with each other. The channel used by users to communicate and exchange thoughts and ideas during the collaboration may assume different types such as text, audio, and video.

To provide group awareness to help users coordinate their work the model has to capture the relationships between users and the tools or artifacts they currently use, i.e. the current collaborative context.

Furthermore, collaboration has to be structured according to the collaborative context. For example, working groups are established when users join the same session. Other users may want to create a new collaborative session to initiate a new group work or may be latecomers, joining an active session. This collaborative session management also must have some kind of security access control to allow only users who have permission to collaborate on a specific task or job.

Compared to single-user applications, collaborative designers do not always want to give to all users full control over the artifact that are being worked by the group. Concurrency control techniques may be necessary to coordinate users and define who is allowed to work, when they work and how they work. Flexible solutions are required to allow fine-grained control over parts of the application and switch of control between different modes according to the activity being made.

To summarize, the following minimal collaborative requirements should be applied to single user applications to make them collaborative (in addition to the already implemented single-user application requirements): communication, group awareness, session management and concurrency control.

It is challenging to design and build a system that supports the requirements mentioned above effectively, especially on single user application that already has many features and a complete set of functionalities for its domain. However, even complex applications follow basic rules of storage and data manipulations, such as several types of data structures and operations that transform and show the data for the users. In general those data structures and applications are extensible and can be modified to support new operations and structures needed to implement the collaborative requirements.

According to Schuckman et al. [11] the step from single-user application development towards groupware development requires more than just sharing common artifacts or connecting a set of distributed user interfaces. Therefore, our proposed mapping allows a uniform handling of the minimal groupware specific aspects: communication, session management, concurrency control, and group awareness on a high abstraction level, which provides a good basis for designer of applications from the conceptual point of view.

Nowadays the MVC architectural style is the most accepted and used architectural style in single-user applications, providing the developers with a common framework for single-user architecture. MVC is a widely used architectural style that separates the data underlying the application (the model) from the input handling code (the controller) and the display maintenance code (the view).

We believe that with specific modifications on the MVC components, multi-user real-time collaborative applications can be derived from this single-user architecture style. For example, the Groove system [4] has shown that the single user MVC model can be modified to include a command processor which intercepts changes to the local model, and sends them to other MVC agents as they happen or when other agents connect.

Starting from a generic MVC-based application, we can assume that a logical a consistent distribution of functional components on the application has been made. For the View part is common to find components that handle the display of data and allow the capture of the user interactions on the Graphical User Interface (GUI). On the Controller part there are components that handle user's data, such as components that validate the user inputs and take care of session data. The Model part usually contains persistence components and customized components that represent specific domain rules. Infrastructure components are spread across all the parts of the architectural style.

The communication requirement may act upon two functional aspects: the communications between the users and the communication between the models of the distributed applications. Theses two functional aspects involve, respectively, components of the View and Model part of the application. Besides these functional aspects, the developers also need to consider the infrastructure necessary to exchange data between the applications.

The user's communications do not change the state of model because it is used only to support the conversation between the users during the collaborative session. In order to support the user's communications the user interface of the applications has to be modified to capture and replay the communications according to the channel and the media used.

The communication between the models of the applications requires a more detailed solution. Since the single-user applications do not expect external changes on the model during the user session, a direct mapping of components cannot be made. However, these external changes are related to the components that handle the model's internal representation.

The modification on the local model, and consequently the changes that are propagated on the remote sites, need to be merged on the local and remote models to maintain consistency among the user's models. The merge is handled by the concurrency control mechanism chosen by the designer and all the details that it carries with it. Again, the single-user application does not have any concurrency control, leaving to the developer to implement this functional aspect or use some pre-built component. The concurrency control mechanisms affect primarily the components that handle the internal representation of the model and the persistence components, mapping the concurrency control mechanisms directly to these components.

The users need to be notified of the external modification made on the model by remote users. The notification can assume the form of an update on the user interface to inform the users of the changes presenting to the users a new functional aspect: group awareness. This functional aspect, as with the user's communications, is directly related to the user interface, mapping the user interface components to multi-user widgets, which is a common approach to support group awareness on applications.

Another aspect of group awareness is the notification of the participation of the users during the collaborative session. A session access control must be created to guarantee that the right users access the right session to do the right task. Most single-users applications do not present a way to authenticate and authorize users but multi-user real-time applications must handle authorizations and authentications not only to control session access but also to inform the other users who are participating on the collaborative session.

The implementation of this session control must not be made entirely on the application. On replicated architectures, a special host is designed to store the data about the current and past collaborative sessions and to concentrate the access of users that wish to participate on collaborative sessions. The server host also stores the most updated model of the current collaboration, allowing late-comers to receive the most recent state of the shared model when they enter in active collaborative session.

From the components perspective it is necessary to create new interfaces to support the new behaviors of the components mapped to the collaborative requirements. These interfaces are based on the desired collaborative functional aspects for which component is responsible. Interfaces for user communication, group awareness and session management are the requirements that must generate new interfaces that must be implemented on the View and Controller components.

The concurrency control collaborative requirement must be implemented on the model components, allowing the concurrent access of the data by multiple remote users. Since most operations supported by the single-user application are based on serialized access, the developers must remove the components of the Control part that do not provide interfaces for the concurrency access and replace them by new components that support concurrency access to data.

New interfaces are not always the best alternative to implement the collaborative requirements. Overwrite existing methods, creating wrappers and adapters for the components and the ability to listen to events triggered by some user action are alternative ways to modify the components without spending too much development effort.

The View components are mapped directly to User Interface controls and Multi-user Widgets, addressing the group awareness requirements. Control and Model components are mapped to components that handle concurrency control mechanisms and Network communication, which is responsible for the model communication, session and concurrency control requirements. The mapped suggested multi-user components can be implemented based on a pre-built component, a framework or a user interface control.

In some applications the components are not as manageable as necessary to be modified or the components do not have enough interfaces to allow change of behavior. In these situations, more radical approaches, such as a complete reprogramming of the components or the replacement of them with components more susceptible to changes, must be worked.

Some functional aspects, such as the concurrency control, cannot be mapped because they do not exist on the single-user applications. In these cases, the most recommended approach is to implement these functional aspects using new components that provide the desired functionality. If some components that possess the desired functionality cannot be found, the developer has no choice other than constructing it from scratch.

The proposed mapping maintains the local model of the users, allowing them to work with and without the collaboration on the same application. On existing groupware applications it is common to find the replicated distributed architecture that allows the users to maintain their local model and the server to have a master copy. Replicated architectures make good usage of network resources because display data are not transmitted over the network: only the control events need to be transmitted.

Since no structural modifications are suggested by the mapping on the MVC architectural style, the replicated distributed architecture is highly recommended to allow the sharing of the model and to handle the collaborative sessions across remote sites. The host server of the replicated architecture, hereafter referred to as the Collaboration Server, is based on simple client/server architecture. It must have a simple security control access to the collaborative sessions and must store the model. It can also be used to implement portions of a distributed concurrency control algorithm.

The proposed mapping raises many technical issues including distributed architectures, concurrency control, view update, networking communications, and so on. These issues must be addressed by the developer considering the resources available to solve them during the development phase.

The goal of our mapping is to provide a generic component-based guide to applications based on the popular model-view-controller (MVC) architectural style. The mapping aims to help designers in the analysis and design discipline during the elaboration phase of the development of groupware. It is important to mention that these guidelines are abstract and conceptual and do not address technical implementation details.

IV. PROOF OF CONCEPT

After reviewing several different tools and technology we have chosen an existing single-user drawing editor called ArgoUML [1] to verify the applicability of the proposed mapping as a proof of concept. This open-source CASE tool provides partial support in UML editing, has a strong community of developers and has received good assessments from the specialized press.

The application has a highly modularized structure and is organized on several subsystems, distributed across the MVC architecture. Every subsystem in ArgoUML has a name and it is implemented as a single directory/Java package which can have a Facade class and or Plug-in interface to allow the use of the subsystem's classes.

ArgoUML uses the MVC architectural style separating the Model, the Control and the View into many related subsystems. The Model is implemented across several packages and does not rely on any other part of ArgoUML. However, it depends on one external component to handle graphs, called Graphic Editing Framework (GEF) [6], which is a separated framework that handles the internal representation of the UML diagram that ArgoUML manipulates.

The Control and View are implemented in subsystems such as the Diagram subsystem or the Explorer subsystem. These subsystems rely on the components of the Swing and AWT libraries and the Java Core Components (JCC).

The main collaborative requirements applied to ArgoUML include a chat tool integrated with the GUI of ArgoUML for communication, a shared diagram workspace that shows the user telepointers and automatic refresh upon changes on the local models for group awareness and a distributed lock technique for concurrency control. All the session control is implemented on the Collaboration Server, which was built based on Java sockets classes, with a simple user/password authentication mechanism. The distributed lock technique is also implemented on the Collaboration Server.

The mapping indicates that the GEF components needed to be extended to allow the multiple editions on its internal elements. The main modifications include code for support external concurrent modifications on the internal hash table and code to make the Collaboration Server modify the internal representation of the diagrams elements.

Three user interface widgets for awareness were implemented: a text box and a button to allow the chat tool, the painting of the diagrams elements to display the lock (green: no lock; red: locked by some user) and a telepointer with the user's name.

To maintain the current implemented features of ArgoUML few the modifications were made on the classes of the ArgoUML. In most situations the original classes were inherited and their functional methods overwritten. Composition was also used in many cases, especially in classes where notifications must be propagated. To organize and apply the necessary modifications and the new code we used many design patterns, such as the Singleton, Observer and Factory Pattern [14], into the original code.

The main technical problems encountered during the implementation of the mapping on of ArgoUML were related to the large number of classes that were modified (the 0.16 version of ArgoUML has more than 1,000 classes) and the lack of documentation about design of the project.

V. Experimentation of the Prototype

In order to evaluate the prototype tool, an experiment was conducted in a controlled environment. The experiment involved twelve students divided into six pairs randomly without repetitions. All the students have a degree on computing courses and their ages range from 23 to 34 (mean: 26; standard deviation: 2.27) with 6 of male sex and 6 female sex.

Each member of the pair was taken to a separated room where he/she could only communicate with his/her pair through the chat embedded on prototype tool. Before the start of the experiment every student received a questionnaire asking about his/her previous knowledge of UML, collaboration technologies and others social aspects. Then the students received tutorials introducing them to the prototype and explaining how to elaborate simple UML class and Use Case diagrams with the help of the prototype.

After this training period the students started the main part of the experiment. Each pair completed three collaborative modeling sessions in which they received a fictitious scenario. They were asked to design collaboratively UML diagrams based on the presented scenarios and using the prototype tool.

At the end of each task the students received the effort perception questionnaire about the task that was completed. At the end of the last task each student participated on a quick interview with the observer.

We are still analyzing the data collected from the experiment but preliminary results indicate that the collaborative models produced by the students with a high level of interaction during the sessions (e.g. more dialogs or high number of elements created/deleted on the diagram) were more complete than those made by students that do not interact much with their partners. The data also suggest that the prototype should be used more often during the modeling of UML diagrams, since the majority of the students involved with the experiment enjoyed the experience and prefer to use a collaborative modeling tool instead of a single-user modeling tool.

Although we do not complete the analysis of the data collected the initial results indicated that the mapping allowed the building a collaborative prototype that satisfy the basic collaborative requirements.

The implications of this experiment can lead to new opportunities not only on real software projects but also on distant learning environments.

VI. Conclusions & Future Work

In this paper we presented an approach to extend single-user application to allow minimal collaborative aspects based on a mapping of the components of a MVC-based application. We showed that, by using the proposed mapping and extending existing components, core functionalities in groupware design can be implemented on new classes of applications that do not support collaboration.

This paper also has shown the implementation of the proposed mapping in a single-user open-source application called ArgoUML, which allows the modeling of UML diagrams. The prototyped application, called CoArgoUML, was used on an experiment to help evaluate if the collaborative requirements were met.

The proposed mapping creates precedent for a guide that helps the developers of open-source applications to implement aspects of collaboration on their projects. The presence of such mapping could encourage the open-source community to extend existing applications therefore increasing the number of domains of groupware applications.

Researches interested on the CoArgoUML prototype can download the version used on the experiment from the web site http://www.comp.ita.br/~pichilia/argo.htm.

Acknowledgment

The authors wish to thanks the anonymous reviewers that made suggestions to significantly improve the presentation of this paper and the individuals involved in the experiment.

References

[1] ArgoUML, *ArgoUML*, <http://argouml.tigris.org/>, July 2006.

[2] J. C. A. Begole, M. B. Rosson, and C. A. Shaffer. “Flexible collaboration transparency: supporting worker independence in replicated application sharing systems”. In *ACM Transactions on Computer-Human Interaction*, pp. 95-132, 1999.

[3] P. Dourish and W. K. Edwards. “A tale of two toolkits: relating infrastructure and use in flexible CSCW toolkits”. In Computer Supported Cooperative Work, vol. 9, pp. 33-51, 2000.

[4] C. A. Ellis, and S. J. Gibbs. “Concurrency control in groupware system”. In *Proc. Of the 1989 ACM SIGMOD International Conference on Management of data (SIGMOD'89)*, pp. 399-407, 1989.

[5] D. Garnkel, B. Welti, and T. Yip. “SharedX: A tool for real-time collaboration”. *HP Journal*, vol. 45, pp. 23-36, 1994.

[6] GEF, *Graph Edition Framework*, <http://gef.tigris.org/>, July 2006.

[7] J. Hill, and C. Gutwin. “The MAUI toolkit: groupware widgets for group awareness”. In *Computer Supported Cooperative Work*, vol. 13, pp. 539-571, 2004.

[8] C. Gutwin, and S. Greenberg. “A Descriptive framework of workspace awareness for teal-time groupware”. In *Computer-Supported Cooperative Work*, vol. 11, pp. 411-446, 2002.

[9] D. Li, and R. Li. “Transparent sharing and interoperation of heterogeneous single-user applications”. In *Proc. of the 8th ACM Conf. on Computer Supported Cooperative Work (CSCW'02)*, pp. 246-255, 2002.

[10] Microsoft Corporation, *Microsoft NetMeeting*, <http://www.microsoft.com/netmeeting>, July 2006.

[11] C. Schuckmann, L. Kirchner, J. Schümmer, and J. M. Haake. “Designing object-oriented synchronous groupware with COAST”. In *Proc. of the 3rd ACM Conf. on Computer Supported Cooperative Work (CSCW'96)*, pp. 30-38, 1996.

[12] Sun Microsystems, Sun Forum 3D, <http://www.sun.com/products-n-solutions/hardware/docs/Software/collaboration_communication/sunforum/sunforum_3d/index.html>, July 2006.

[13] S. Xia, D. Sun, C. Sun, D. Chen, and H. Shen. “Leveraging single-user applications for multi-user collaboration: the CoWord approach”. In *Proc. of the 9th ACM Conf. on Computer Supported Cooperative Work (CSCW'04)*, pp. 162-171, 2004.

[14] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1994.