Adaptation of Single-user Multi-touch Components to Support Synchronous Mobile Collaboration

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

Abstract - **Mobile applications, which include calendars, browsers, and text editors, are part of our lives nowadays. Most of the mobile applications are single user, i.e. they do not allow the cooperative work of group of users simultaneously. Adaptation is a technique to transform single user application into cooperative one. A form of adaptation is based on the reuse of the manufacturers' SDKs (Software Development Kits). In general, adaptation is made manually; nonetheless, we conjecture that some automation is possible. This paper investigates this possibility for mobile applications by assuming that the target SDK's components comply with certain design guidelines. We present a reference model to develop collaborative mobile groupware applications by the modification of multi-touch user interface components provided by the manufacturers' SDKs. We also present a process to convert and adapt those components. We discuss two examples that illustrate our proposal.**

Keywords: **multi-touch, mobile, collaboration.**

I. INTRODUCTION

Device manufacturers, dotcom companies, and telecommunications carriers have provided many resources and business opportunities that motivate developers to create, publish, and distribute mobile applications. The current communication and computation infra-structure for mobile devices enable the collaborative work; i.e. users working together on tasks and pursing a common goal at distinct places. Real-time collaboration occurs when the users perform simultaneously some cooperative work, such as synchronous editing. The support for real-time collaboration that allows interaction among geographically distributed participants at any place is still a considerable software engineering challenge due to the effort required to support properly the collaborative requirements. Collaborative requirements include low communication latency, awareness widgets, coordination mechanisms, concurrency control techniques and others.

The mobile collaboration's environment presents a broad range of use of technology, novel social practices and behaviors, and has potential for the exploration of the different roles that cooperation, communication, and coordination provide. Our work concentrates on the technical aspects of the design, architecture, and organization of the mobile application's components.

The choice to focus on multi-touch components is due to the fact that they are the most employed interface components nowadays. Another reason is that previous research efforts show that they allow user interactions more suitable to synchronous collaboration.

Although the current SDKs (Software Development Kits) provided by the two most popular mobile platforms, Android and iOS, contain many components organized as standard application architecture, the development of groupware in mobile context is a challenging activity as it involves the understanding of both technical and social factors. Some research strategies to support collaborative features in mobile applications include toolkits, Transparent Adaptation, and Component-based development. These approaches aim to reduce the design effort, but none of them focuses directly on the automatic modification of the component's source code organized in a specific architecture style.

We propose an extension to an abstract component model for developing mobile groupware applications based on the modification of the user-interface components. In order to guide the developer that is adapting the application we propose a process to construct prototypes of mobile applications with collaborative requirements.

The rest of the paper is organized as follows. The next section describes the existing approaches to create collaborative applications. In section 3, we propose an extension of the abstraction model, based on MVC, for groupware application development, its technical characteristics, design, and architecture. In section 4, the component modification process is presented with two examples of adaptation using the proposed process. The examples aim to verify that our approach is sound. Section 5 presents an evaluation of our proposal. Finally, we conclude with a summary and directions for future work.

II. RELATED WORK

The traditional research strategies to support collaborative features into existing applications are based on the classical techniques *ad hoc* modification [1, 9, 17, 18], Toolkits [3], Transparent Adaptation [6], and Component-based development (CBD). A technical comparison of these approaches is presented by Pichiliani and Hirata [12].

The *ad hoc* modification, Toolkits, and the Transparent Adaptation approaches reduce the design and development effort, but they do not provide systematic features for the automatic modification of the component's source code organized in a specific architecture style. Also, they demand considerable development and design efforts that result in restricted collaboration styles.

The CBD approaches have been the focus of many research efforts from the replacement of components [4] to new

COLLABORATECOM 2012, October 14-17, Pittsburgh, United States
Copyright © 2012 ICST
DOI 10.4108/icst.collaboratecom.2012.250449

frameworks [15], plug-ins [13] and architectures [5, 16]. They represent the state of the art software engineering techniques that alleviate the effort demanded for the development of mobile applications.

Among the existing CBD approaches that address the modification of legacy applications to support collaborative features, the EXEC Framework [7] is an abstract model for groupware applications and also a semi-automatic transformation tool that converts existing components to support collaborative requirements. The approach focuses on components structured according the MVC (Model-View-Controller) architectural style. 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).

Although all the aforementioned approaches provide a gain in terms of development effort, in general, they do not suffice to meet the collaborative requirements due to the high level of customization required by them. Therefore, some level of *ad hoc* modifications of the source code to change the services provided by the components according to the collaborative requirements is required.

III. THE MULTI-USER MVC MODEL

This section describes the Multi-user MVC Component Model followed by the presentation of the abstractions, simplifications, and modeling elements based on the existing Shared Component Model [7]. We focus on the characteristics of the model including global identification, interception point, coordination services, composite structures, and property changes. The following subsection comments the specific details of the model implementation on the Android and iOS platforms. Then the aspects of the component's design are presented from the perspective of the features they should support. Finally, the characteristics of synchronous collaboration infrastructure are described from the replicated architectural standpoint.

A. The Model

We propose an extension and some adaptations of the abstract collaboration model from groupware applications presented by Li et al. [7]. The extension focuses on the mobile platform and the MVC architectural style. In some aspects our model reuses and combines many features and strategies employed by the approaches described in the previous section.

The reasons to choose MVC are two-fold. MVC is the most accepted architectural style to organize the components of mobile applications. MVC also provides the developers with a common framework for single-user architecture. This is corroborated by fact that the current mobile's SDKs recommend that developers organize the structure of the applications and their components according to MVC.

Applications built with the MVC architectural style are split into three parts: Controller(s) responsible for input handling, View(s) responsible for output, and Model that implements the underlying application and data. Controller(s) translates user inputs into updates to the Model state. When the state is modified, Model notifies View(s) that view updates may be necessary. View(s) then recomputes what (if any) display updates must be made. This architecture style frees Model from details of how View(s) are updated, and frees Controller(s) from having to determine which View(s) must be modified as a result of user inputs.

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

Fig. 1 shows the traditional single-user MVC architecture style and our reference model with the MVC parts mapped inside the original Shared Components Model. In this extension, named Multi-user MVC Model, the groupware application contains GUI elements, which display replicated data for each user by the multi-touch and multi-user interface components contained in View(s). The data flow occurs by the interaction of the user with multi-touch controls such as buttons, text boxes, radio buttons, maps and others that contain data object structures. The multi-user aspects of these controls require changes and modifications on the Controller(s) and View(s) parts, thus the components are grouped as the Modified Collaborative Components label shown in Fig. 1.

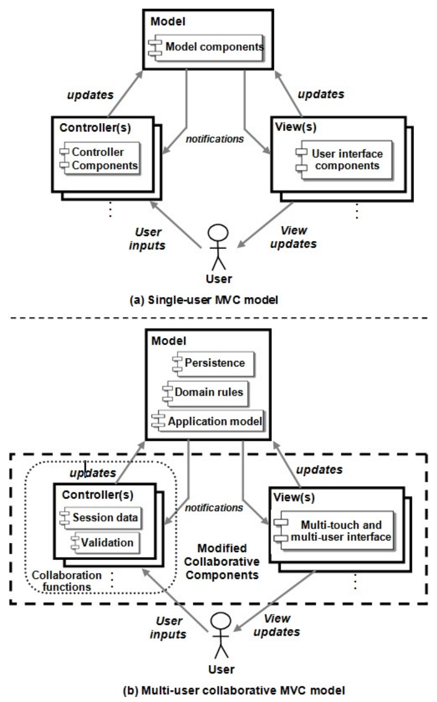

Figure 1. (a) Single-user MVC and (b) the Multi-user MVC model (b). In (b) the MVC is mapped over the original component model proposed by Li et al.[7].

The collaborative components in our model are found in the Views and the Controllers parts, since mobile applications separate their features in more than one pair of View/Controller. If we model the components of the Model part as Persistence, Domain rules and Application Model our reference model must not consider them as components that are involved directly in the collaboration because they do not contain the original data objects handled by the user-interface controls, which are the main data that the model aims to share.

As in the original model, we also consider a runtime system that dynamically provides access to services such as event interception and execution, events broadcasting and notification, data replication and persistence, session management and network communication. We model this runtime system, named Collaboration functions, as a group that encloses the Controller(s) of the application.

One important aspect that must be taken into consideration is the fact that most multi-touch components do not separate the data and the control with well-defined interfaces as required by the original abstract component model. However, they provide means to obtain the data that must be replicated and the mechanisms to modify how the component interacts with the user through events, methods and properties.

To better understand how the local communications flow between the components let us consider a graph editor application as an example. This application has a View that contains user interfaces to work with the edges and vertices of the graph being edited. The Model contains a data structure that stores the graph's edges and vertices along with other information. Controller handles events that perform actions such as the creation, update or deletion of edges and vertices. When the user wants to create a new edge, she interacts with the interface of View that produces an event to be handled by Controller. Controller sends a message to Model that updates the internal data structure that represents the graph. Model also notifies Controller and View that a new edge is persisted and can be visually presented on the user interface. A similar interaction between the parts occurs when an element (vertices or edge) is changed or excluded upon user's commands in the application interface. This example describes a local processing that happens between the MVC parts.

Our model represents the components of the Model part by its functional features, i.e. Persistence, Domain Rules and Application Model. These components contain core programming logic and inner working algorithms required for the correct data manipulation according to the application requirements. Unlike the traditional approaches, our proposed model considers that the replication of the user interface data are handled directly by the Controller(s) components and the Model components are not involved for data synchronization. This approach may lead to a number of conceptual and technical problems such as data that is not represented in user interface components or the synchronization of applications that adapt GUIs on different devices.

We consider that the only data demanded to be replicated by our pre-defined collaboration style is held at Controller(s) and View(s) and not in the Model part. For instance, suppose that in the graph editing application, Model contains a component with the following domain rule: the value for a property associated with an edge cannot be less than a specific value computed from an internal algorithm, such as the logic employed for fluid flow control analysis on graphs. This rule is coded inside the Model's components and its logic and internal data do not need to be replicated since only the original value stored with the edge is relevant for our collaboration style. The synchronization of the value stored in the edge of the graph and collected by a GUI control reflects the data change to the Model components of the other users in the collaboration, thus the same data value is sent to the Model at a remote site that performs the correct domain rule execution. Furthermore, the data that is not stored within the properties of the user interface components is not an issue according to the defined collaboration style that our model and process focus on.

The synchronization of data for GUI elements that depend on specific characteristics of the device is not addressed directly by our model since this synchronization is handled by the concurrent control mechanism chosen for each UI element. The fact that the same control is represented in different ways for each device involves the aspect of dynamic generated GUI controls. This aspect is a technical issue that our automated modification process handles by scanning, analyzing, and considering the possible ways that dynamic GUI controls are rendered in the user interface of the device when the application is being executed. Also, our modification process, as described in the next session, contains a repository of control's template that catalogs required event modifications, mapping of properties, and behavior of the user interface elements separated by each variation of device for the same platform that can occur when the same application is adapted for different devices. At the worst case scenario, when the automated modification process cannot detect the creation of dynamic GUI components, the process highlights the source code area to be manually changed and provides means to implement *ad hoc* modifications on the main events of Controller(s) that handles data manipulation for dynamic generated GUI controls.

B. Required Data Object Characteristics

One aspect that must be detailed in our model is how the data object structures of the user-interface components are accessed, replicated and modified to support synchronous collaboration. The model addresses the data object replication aspects by describing five required characteristics in the component model referred to as global identification, interception point, coordination services, composite structures, and property changes.

The global identification characteristic deals with changes in the property of components that must be eventually applied in all replicas to maintain consistency across cooperating sites. Therefore, any distributed environment infrastructure must be able to uniquely identify divergent instances of the same shared data component.

Since we deal with a well-structured application with components already separated by the MVC, the global identification characteristic is maintained in our abstraction and we extend it further by modeling that besides the component global identification we also must be able to globally identify the component by a full description of the container (i.e. the specific View or Controller), the application device, the platform, the collaborative session and the users. To achieve this goal we propose a fully-qualified global identification that allows the developer to rapid access all the path for reaching the required data. For instance, a device *D* that runs on the platform *P* is executing the application *A* that is in the collaborative session *S* when the user *U* changes the property *R* with the value *E* of the component *C* stored in the View *V*. The complete fully-qualified global identification for this scenario could be accessible by the developer in the form of a suitable object notation similar to *D.P.A.S.U.V.C.R.E.*

The interception point characteristic is seen as a mechanism supplied by the component to allow read and write access of the properties values in order to reflect the user interaction with the GUI. It is a common practice to implement coding techniques that creates a hooking point allowing the interceptions of properties changes before and after they take the desired effect that the component perform on the data.

One specific aspect that originates from the MVC architecture style concentrates on the mechanisms for communication of data and events from the three parts of the application. Our modeling builds upon these mechanisms in such a way that data are sent though updates, notifications, and requests already performed between the components of the Model, View(s) and Controller(s) parts of the application.

The coordination services are the mechanisms responsible for the implementation of different techniques involved in coordination of work according to the data and the adopted style of coordination, i.e. pessimistic or optimistic concurrency control techniques. We keep our model flexible enough to support the coordination services similar to the ones presented in the components provided by the Flexible JAMM [4] and we also follow the design recommendations proposed by the design of an API as in the Transparent Adaptation approach [8].

An object representing some user-interface component may recursively contain many other objects with different structures and requires its own way of expressing the relationships between the internal objects. This is especially true for multi-touch components that do not have a clean and simple separation of data and control as indicated by Yang and Li [19].

Some standard components available to the developers in the SDK have to follow specific rules to communicate their data through the internal MVC component's structure of the control, thus increasing the design and developer effort to read and write the data inside the underlying object structure. For instance, a visual component that allows the selection of an item in a list demands some lines of code to connect the View part of this control to the Model part and also the manual assignment of the control's instance to a variable that allows the programmer to access the data object structure via properties. However, the source code is available and the application is already built so all the required coding infrastructure for the component is ready and can be accessed by the recommended public interface and mechanisms provided by the SDK.

The property changes characteristic focuses on distinguishing the type of property that a component possesses, i.e. the components have properties that allow the replacement of an old value with a new one or properties that allow incremental changes. This abstraction is based on the fact that the mechanisms for replication and concurrency must know the interface so they are aware of how properties work and how to handle different types of data reading and writing.

Unlike the original approach, our model does not require differentiation of properties since we do have the source code and most user interface components we focus on communicate of the properties changes in organized and systematic ways, which are implemented with well-known techniques that glue the parts of Model and View thus abstracting the details to obtain and change property values.

C. SDK Specific Details

Nowadays the two main platforms for mobile development are Android and iOS. In order to understand how to employ the Multi-user MVC Model and the automatic component conversion tool one must first understand how the SDK of those platforms organize the application and its components.

The implementation of a new mobile application begin from a starting point that is usually a new empty project based on an existing template provided by the IDE (Integrated Development Environment) that comes with the SDK. While the documentation and the official development guidelines recommend this step, many developers create their own custom project with specific organization of the components or instantiate a third-party framework. In these scenarios our automatic conversion process is not able to automatic recognize the required components and modify them. Therefore, a pre-condition to implement our model and employ the automatic component conversion process, described in the next session, is to follow the standard development practices and use the library provided by the SDK for default application creation.

Both Android and iOS platforms structure the application in the MVC style. In general, they rely on XML files that contain the user-interface components definitions represented as tags that allow the customization of the visual and functional properties of the component. To address the dynamic aspects it is required to link the components to a section in the source code where the developer can program the components' behavior. The Views are represented by XML files that are connected to a specific class of the project allowing the developer to access all the components by the source code of the class that corresponds to the Controller. The Model part is implemented exclusively with other source code files of the project.

The Android platform uses the Java language and has the concept of *Activity* that corresponds to a View representing a single screen within the user interface. For example, an email application might have three activities: one to show a list of new emails, one to compose an email, and one to read emails. Although the activities work together to form a cohesive user experience in the email application, each one is independent of the others. As such, a different application can start any of these activities (if the user has the correct access rights). For example, a camera application can start the activity in the email application that composes new mail, so that the user can share a picture via email.

The applications developed for the iOS platform rely on the Objective-C language and have a clear separation between the GUI (the View) and the actual code that provides the application logic (the Controller). In general, each View has a ViewController class behind it that reacts to user-interface events such as button presses, table row selection, or tilting the device. As in the Android platform, the View and the components are defined in a XML file called *nib*, whereby the developer creates the description of the GUI he/she is building. The developer also needs to connect the *nib* file to the ViewController class allowing hooking points to handle the event's behavior of the View elements.

Although our approach focuses on the implementation of synchronous collaboration features in existing applications for the Android and iOS platforms, our model and process are not limited by the UI elements provided by the SKDs/IDEs for mobile development. By performing some adaptations it is possible to extend the abstractions, modeling elements and the systematic modification process to other contexts where the synchronous collaboration features may benefit the group work. Here we concentrate our efforts on the mobile platforms due to the large established database of applications found on the online app stores, the exploration to provide social aspects in mobile scenarios, and collaboration opportunities to not only support traditional cooperation, communication, and coordination requirements but also promote novel approaches to those collaborative features.

D. Collaborative Component Design

The multi-user version of the user interface components are based on the existing controls of the GUI provided by the SDK and available to mobile application developers. The modifications to make them collaboration-aware should be based on a pre-defined collaboration style in order to simplify and reduce the design and development effort required to create prototype applications with simple collaborative features restricted to the user-interface controls.

Traditionally work on Transparent Adaptation and other CTS (Collaboration Transparent Systems) [17] focuses on applications that allow the editing of documents, most notably the collaborative editing of text by transforming and converting single-user editors to multi-user applications. The Flexible JAMM [4] is an exception since it provides a complete network infrastructure and replacement classes to switch standard controls for collaborative ones in the Java platform whereby the components of the graphic library Swing are employed. Here we follow this approach in the sense that our modified controls present a pre-defined collaboration style that is implemented automatically in the development phase by direct modification of the source code.

Following the Flexible JAMM approach, the system must allow collaborators to work together closely or independently. To that end, the system should support the features that include: (i) Simultaneous work when desirable; (ii) Use implicit floor control as the default, and allow explicit control when required; (iii) Location-relaxed WYSIWIS; and (iv) General group awareness information.

The component design also should maintain all the existing features to keep the user expertise and experience with the application. To illustrate the final behavior of the components suppose we have a simple mobile application that catalog books read by the user. This application contains two Views: one for listing the books already read and another View for inserting and editing book information. Both Views contain standard controls such as buttons, text boxes, date pickers and a list that shows the books as items that can be selected.

In our approach the main user interface components must retain its existing features if the user does not want to collaborate, thus keeping the application the same. If the user starts a collaborative session the controls must be collaboration aware. In this scenario the text boxes must allow multi-user editing, the buttons must allow any user in the session to press them and the list should allow local and remote users to select items. The specific awareness, group information, concurrency control and relaxed WYSIWIS are available to each control in a standard setting, i.e. already defined, but it is possible to change these settings in execution time.

The standard settings for the collaboration controls are based on the multi-users features of Flexible JAMM. However, we do not provide specific awareness widgets such as radar views or telepointers, since our goal is to keep the user experience with the application that the users already have. Also, the insertions of such awareness widgets require layout and aesthetic considerations that could not be suitable to existing mobile applications, thus specific design is recommended in order to accommodate those widgets. Nonetheless, since we do change the source code directly, our approach still keeps the option to manually change the appearance, behavior and other aspects according to the developer and designer requirements.

Multi-touch controls can provide new styles of interaction when they allow more than one user at the same time. These controls represent possibilities to interact and perform gestures together by considering local and remote gestures in the interface that can increase the awareness and yield a further sense of collaboration. For instance, the traditional pinch or spread gestured required for zooming out and zooming in an image, respectively, can now be performed by a pair of users that must coordinate and collaborate to reach the desired visualization effect on the image. While we believe that this scenario can confuse users at first, the collaboration at the control level for mobile applications presents interesting possibilities that already have been explored in others contexts such as table-touch interactive surfaces [11].

E. Synchronous Collaboration Infrastructure

The implementation of the requirements that share the object data to adapt existing controls for multi-user interaction is addressed in our model by a runtime system that contains features for event handling (interception, execution, notification, and broadcast), data replication, concurrency control, serialization and deserialization of objects, and session management. The source code of this runtime layer is automatically injected in the application and it is the synchronous collaborative infrastructure needed to perform all the implicit data communication among the participants of the same collaborative session.

The replicated distributed architecture is highly recommended to foward the actions that change the Model and to handle the collaborative sessions across remote sites since no structural modifications are required to the MVC architectural style. The host server of the replicated architecture, hereafter referred to as the Collaboration Server, is based on a client/server architecture hosted in a server computer. It encapsulates from the developer all the technical details involved in sharing the components and their property changes by employing additional support from the environment that the components lives in.

Fig. 2 shows the distributed replicated architecture of our runtime infrastructure where two users are communicating with the Collaboration Server by sending local changes in their Models to the Master model and receiving notifications from the server's Model. The runtime infrastructure wraps the Controller(s) and View(s) parts of the application and it is the software layer that provides communication through the network between the local and remote devices with the Collaboration Server.

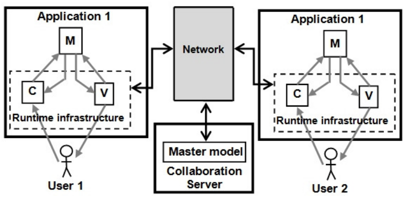

Figure 2. Replicated architectural view for two mobile users with a modified MVC application collaborating over a network.

The Master model is created from a subset of the main classes and components found on the Model part of the application. This subset is abstract enough to contain the property changes that must be forwarded to all participants and handle data replicas, concurrency control data structures and operations, session management, and security features. The development of the Master model must be guided by the possible collaboration styles that the multi-user version of the application is based on, including how to support the coordination and cooperation requirements. Traditional technical challenges of the mobile platform, such as connectivity management, *ad hoc* routing, dynamic service discovery, limited bandwidth, unreliable synchronicity among the devices, and network communication in heterogeneous networks also are addressed by the Master model.

The Master model is dependent of the local Models in a sense that it forward and exchange data instead of centralizing all the data/state inside a single Model. This approach may cause a lack of synchronization among the local and remote collaborating users, but it provides a flexible operation of the application when they are not collaborating. Only a set of the Model features found on the application is implemented in the Masters mode instead of the duplication of the entire application's Model.

From the components' perspective the fully-qualified global identification *id* is assigned when the object that represents the user-interface control is instantiated. Next, the concurrent control mechanism chosen decides how to apply property changes on all data replicas. Then the property changes are pushed over the network to be applied on the modified collaborative components. Due to the fact that property changes are intercepted by the runtime infrastructure at distinct event levels, e.g. key pressed, mouse moved, and tilted device events, both pessimistic and optimistic concurrency control can be implemented. The runtime infrastructure is responsible to accommodate late-joining users and synchronize their sessions by using a combination of the direct state transfer and event history replay techniques [6].

By employing a replicated architecture with a Master model that is updated when the local modifications are broadcasted by the collaborating users it is possible that some participants may lose the synchrony between their local Models and the Master model at a given moment, thus generating an inconsistent collaboration state. This effect can be mitigated by the concurrency control mechanism employed and by the use of a social protocol that supports the negotiation of actions through the communications channels found in the mobile devices.

IV. THE PROCESS FOR MODIFICATION AND ADAPTATION

The process that converts and adapts the user-interfaces multi-touch components of an existing mobile application in order to support collaboration requires that the specification of how the collaboration takes form must be already well defined by the developers. This is valid for almost every automatic process that performs actions when the goal is to alleviate the developer's work by reducing the development time and effort. The amount of automation that can be performed, especially those that involve generation of source code, can be seen as a function of how accurate the specification reflects the requirements that are addressed by the application's features. We stress that even with a pre-defined collaboration style and the options available to customize the collaboration behavior, the specification and the needed expertise on the domain must be anticipated and obtained before any automatic or manual developer action for the implementation of collaborative features is carried out.

Starting from our requirements regarding the availability of the source code and the organization of the application according to the MVC architectural style, we develop a process, named MVC UI Component Modification Process, which specifies the sequence of activities required to apply the transformation in the interface components. By following the activities of the process, the developer can reuse applications saving both time and effort when creating prototypes that allow synchronous collaboration.

The adaptation process is depicted in Fig. 3 as an UML's activity diagram where each activity is linked to commentaries that describe its inputs and outputs. In the following, we detail the activities from the perspective of the developer that want to adapt existing projects. In order to describe and facilitate the understanding of how to apply the process, the inputs and outputs of each activity are detailed by applying it to a simple case study mobile application that catalog books read by the user. This application contains two Views: one for listing the books already read and another View for inserting and editing book information.

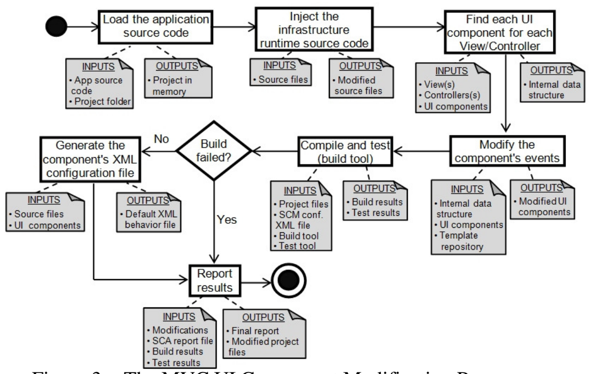

Figure 3. The MVC UI Component Modification Process.

The process starts by loading the application source code from the project files. Since we aim to convert existing applications we suppose that they have already been created by the recommended IDE (Integrated Development Environment) provided by the platform, so we take advantage of the existing structure and load all the required source codes, resources, XML configuration files, manifests, project, and other files that compose the entire project. This activity only requires that the user informs the path that contains the files created by the IDE during the original development phase. The platform (Android or iOS) identification of the project is performed at this stage, therefore automatically preparing the further activities of the process accordingly.

Our book catalog application is based on the Android platform and was developed using the Eclipse IDE. This application was created by the standard mobile application template that automatically created a project root folder with subfolders named *assets*, *bin*, *gen*, *res* and *src* to store the application resources. In this first activity we must provide the project root folder as an input and expect the loaded project in memory as output.

The next activity injects the source code files of the infrastructure demanded for remote collaborative features with all the programming elements (classes, methods, events, delegates, and others) needed to exchange the data with remote applications and the Collaboration Server. This activity also injects a new menu option and a View/Controller pair that allows the configuration of network-related authentication and collaboration aspects.

The injection of the infrastructure runtime source code does not require any user input besides the source file that already has been provided in the previous activity. This step finds the application's main Activity, adds a new option in the menu bar of the application (or creates a new menu bar if it does not exists), inserts a new activity (a View), and injects two source code files responsible to handle network communication.

Immediately after the injection of the infrastructure runtime layer, the next activity of the process inspects the project files, finds each interactive user-interface multi-touch control in each view and marks them to be modified latter. This inspection activity identifies the View, the Controller, the objects that represent the control, the events and also other elements such as its scope, visibility and visualization properties in order to select only the GUI elements that need adaptation for collaboration. This activity does not require any user input and produces an internal data structure filled with locations of the View and GUI components that are used in the next activity.

The actual modification on the selected GUI components is performed in the following activity of the process. The modifications for each type of component are stored in a template repository that contains the required event modifications that are injected directly in the source code in a control-by-control basis. Again, the modifications provide a pre-existing collaboration style according to each control, but it is possible to customize and configure later on the behavior of the controls such as a specific concurrency control technique, the amount of replicated events and other control interactions that affect collaborative aspects.

This activity automatically modifies the delete button in the list contained in the first activity and modifies all the text boxes, buttons, sliders and date time pickers in the book editing View. The definition of how to modify these components is stored in the events repository provided by the process and does not require any user input. The output of this activity is the changed source code files that represent the Control part of the GUI components.

The following activity is the one that does the compilation of the project. This activity relies on external tools for performing automatic builds and also executes any test suite that the developer created. The necessary commands to compile and build the application should be stored in a XML configuration file or provided by the user in the form of command-line parameter since the tool that materializes the entire process accepts this form of execution. If the build fails, the process flow is redirected to the activity that shows to the user the report with compilation errors including detailed information about the problems in the compilation (compiler error, line numbers, callstacks etc), step by step guides to manually implement the needed modifications for synchronous collaboration, recommendations for refactoring, and suggestions provided by static code analysis tools. If the build succeeds the next activity that generates a XML configuration file is performed.

The compilation activity requires a XML file as input for the automatic build and test process. The result is a text file that contains the building and testing results with a flag that indicates if the build process returns successfully or not.

The next activity creates a XML configuration file for the entire project that describes all the automatically modified GUI components and their default values for the control's behavior. This file is stored within the application project and at the first collaborative session it is uploaded to the Collaboration Server. With the configuration of the components stored in the Collaboration Server the developers can change some aspects of the component by modifying behaviors in execution time without redoing all the process activities again. However, if the developer changes the user interface the entire process must be followed from the first activity.

No manual input is required in this activity and a file named *CTRL.XML* is generated with the default collaboration setting for each control modified, which is the default locking mechanism for the text boxes, buttons, sliders and date time pickers of the book catalog application. This file represents the output of the process and is stored in the project's folder.

Finally, the MVC UI Component Modification Process finishes by providing a report containing detailed information covering the modifications performed, the changed and included files in the project, and the compile and tests results. This final activity presents a summary to the developers that can further inspect and modify the source code files in order to check or review exactly what and where the modifications on the application were automatically performed.

V. Evaluation

We manually modified two existing applications according to our process in order to evaluate our model and the MVC UI Component Modification Process. In the following sub-sections, we discuss how we followed the process to change the applications, how some components behave with synchronous collaboration, the level of flexibility that is achieved in our work, and how our approach compares with techniques for modification of existing applications.

A. *The CoMathDoku and CoFingerPaint prototypes*

The creation of the first prototype was based on a popular open source Sudoku application for the Android platform named MathDoku [10] that was renamed as CoMathSudoku after the modification of its components. This application is based on the same rules as KenKen game and is composed of two main activities: (i) *MainActivity* which dynamically creates textboxes for the game; and (ii) *OptionsActivity* which allows the modifications of the game options. The GUI elements are the input textboxes components used to fill the empty spaces with correct numbers required to solve the puzzle.

The first step of the process, *load the application source code*, is trivial since we are using the Eclipse IDE to change the application's project. Then, the execution of the second activity of the process, *injection of the infrastructure runtime source code*, is manually made by importing two new classes that contain the synchronous collaboration infrastructure components to support the collaboration.

Following our process the next activity involves the identification of the View, Controller and GUI components. First, the identification and modification of the textbox controls demanded the inspection and understanding of the source code since these controls were created dynamically depending of the difficulty level of the game, i.e. harder levels created more controls in a bigger puzzle. Second, each time the user touches the control it executes specific programming logic in the *OnSelect* event to notify the interface that the user is editing the selected place. Third, the replicated data needed to be carefully handled to simulate a remote data entry without changing the focus of the control that the local user is currently positioned.

The *modification of the component's events*, the following activity of the process, demanded the replication of the input on the textboxes controls of the user interface. The modifications in the Controller part assume the form of some lines of codes inserted into the *OnSelect* and *TouchStart* events of the dynamically created text boxes stored in the *MainActivity* class. The code inserted is responsible to assign a global identification *id* for each control, collect, and transmit the data value to the other participants in the same collaborative session. This modification is simple once the exact places, i.e. the control's events, needed to be changed are found. We also changed the Controller to receive the data from other participants and call new methods that redirect the execution flow to the local *OnSelect* and *TouchStart* events.

Once the modification in the Controller part for a single text box was made we reused this control in any variation of the game such as new levels of difficulty and other game play configurations. The *compilation and build* activity was performed directly from the Eclipse IDE that successfully generated the files necessary to deploy it to a device. Then we manually performed the last two activities of the process by creating a XML configuration file for the text box and reviewing the modified project files.

From this prototype we learn that a single control's modification can provide enough flexibility and reusability for the user interface sufficient to cover variations of the game. The manual creation of this first prototype required the modification of one class, six methods, and one XML configuration file. Two new classes were inserted to support the network communication and synchronization infra-structure. The overall man-hour effort was 16 hours performed by a senior Java programmer that spent more than half of the time analyzing the application before the coding phase. Fig. 4 shows the result of our CoMathDoku prototype where two users are playing the game.

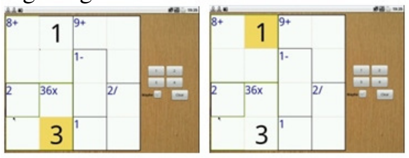

Figure 4. Two users playing a CoMathDoku game simultaneously.

The second example is the drawing application named FingerPaint provided by the SDK samples of the Android Platform [2]. This simple application allows the user to freely draw using touch and the traditional drag and draw action performed directly in the drawing canvas area of the application. The first two activities of the process were performed in the exactly same way as the previous CoMathDoku example.

To find the View, Controller and GUI components we explored the application and found that it is composed of a single view named *FingerPaintActivity*. The touch interactions are handled by the Controller in the *touch_start*, *touch_move*, and *touch_up* events which perform graphics directives to draw according to the coordinates of the touch. The main user interface control in this application is a Canvas object that act as a drawing area and responds to touch events.

The modifications required the localization and modification of the programming block for the touch events in the GUI control that actually performs the drawing. Since this a free drawing application and there is no elaborated selection or concurrency actions that could affect user interaction during the drawing, we did not employ any concurrency control technique to coordinate the user actions. The build, generation of the XML configuration file, and report generation activities of the process were performed manually without further difficulty since this is a simple example application which goal is teach new Android developers how to handle touch events.

The manual creation of this second prototype required the modifications in one class, five methods, and one XML configuration file. Again, two new classes were inserted to support the network communication and synchronization infra-structure. The overall man-hour effort was 10 hours performed by a senior Java programmer. Fig. 5 shows two users drawing together with the CoFingerPaint prototype.

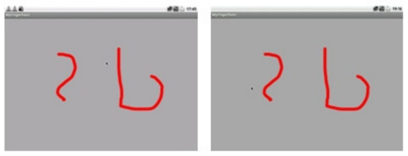

Figure 5. Two users drawing together with CoFingerPaint.

Overall, the modifications demanded in this prototype were classified as simple and were implemented with little effort. The implementation of the process was performed manually in order to evaluate the usefulness of the process and learn key technical issues before turning it into an automatic process that require little manual adaptation when modifying existing mobile applications to support synchronous collaborative requirements. The two applications chosen for the adaptation, MathDoku and FingerPaint, may be simple and did not pose significant design and programming challenge for the proposed modifications, but they represent well the characteristics the of applications found on the online app stores and provided valuable insights and relevant programming experience to show the results of this research.

B. Comparison of the Approaches

The comparison of our approach with other strategies, such as *ad hoc* modification, toolkits, Transparent Adaption and Component-based Development, must consider aspects that include effort, feasibility, and assessment of the convenience of that adaptation.

Ad hoc modifications do not provide an application reference model, systematic code modification, and the organization of components in an architecture style. The Transparent Adaptation do not change the source code directly, therefore the implementation of collaboration requirements are rather simplistic and focused on screen-sharing. In a sense, the characteristics of our approach shares more aspects with the original shareable model proposed by Li et al. [7] than other strategies due to the details and elaboration of our components' modification, automation level, and organization. Therefore, the comparison concentrates in these two strategies.

There are some key conceptual aspects that differentiate our model from the original Shareable Component Model. First, we focus only on the data and control features of the GUI components instead of all the application's components. Second, the MVC architectural style is mapped onto the shared components thus increasing the semantic of the model. Third, we assume that we have the source code of an existing application so it is possible to access all the definitions of the components such as events, properties, interfaces, data structures and internal members. Fourth, our reference model is abstract enough to represent multi-touch mobile applications without being restricted to a set of component models, a platform or a specific technique to reuse the components.

As with any abstract component model, our approach imposes restrictions that define which applications can and cannot be adapted to support synchronous collaboration requirements. These restrictions are represented by the abstractions and modeling elements employed that include organization of components, architectural style, and synchronization of GUI control's data. These restrictions must be met by the applications. Also, the class of the mobile application that can fit into our approach is a subset of the general application model represented by the MVC pattern and that also corresponds to applications eligible for the original Shared Component Model.

The feasibility and convenience of the adaptation must be evaluated by the comparison of the resources and effort demanded to adapt legacy applications. Nowadays the rapid expanding growth of applications found in online app stores imply that the time to market should be reduced as much as possible in order to make the responsible for the development of a new app a competitive player in this overgrowing market. Therefore, techniques that alleviates and reduces the development effort of collaborative features that aggregate social aspects in already developed and deployed products may provide a convenient resource for the development phase when a prototype must be created in timely manner.

Although the argument that the complete re-design of the single user application to support new collaboration requirements may be a suitable approach, we believe that re-design and start-from-scratch actions require more effort, resources, and time than adaptation techniques. This belief is supported by our initial evaluation and analysis of previous research. Further experimental procedures must be conducted to gather in the field development effort data to validate this conjecture.

Other aspects regarding the collaborative environment, such as user privacy, data security, adoption, social behavior, among others represent important factors that influence the insertion of collaborative features on existing mobile applications. Although these issues have a relevant impact from a collaborative adoption and use perspective, they must be evaluated and planned as in any development processes that focus on groupware applications.

VI. CONCLUSIONS & FUTURE WORK

According to Schuckman et al. [14], 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, in this paper we presented a model and an adaptation process that allows a uniform handling of the groupware specific aspects on a high abstraction level, which provides a valuable resource for developers and designers that wish to consider prototype collaborative features in their existing mobile applications.

The MVC Multi-user Component Model presented abstractions, simplifications and modeling elements from the context of MVC applications built for mobile scenarios. Our model is based on the Shared Component Model and details the global identification, interception point, coordination services, composite structures, and shared property changes characteristics. The aspects related to the details of the synchronous collaborative infrastructure originated from the replicated architecture employed were discussed.

The implementation of the synchronous collaborative features take the form of a process that is defined as a sequence of activities that cover all the operations that load, modify, compile and customize the source code of the application's component that is going through the activities of the process.

The reusability, flexibility and feasibility of implementation for the modification of the applications according to our proposed model and process were evaluated in two prototypes focusing on the analysis of its components, internal structures, and programming logic that compose the original applications. Our findings indicated that the model and the adaptation process can assist developers during the implementation of synchronous collaborative features when the goal is create prototype applications as a proof of concept of synchronous collaboration coupled with existing features.

Although our reference model and adaptation process are implemented by several known techniques, they provide a novel approach that allow automated development in mobile contexts and collaborative features.

Future work includes the development of an automatic conversion tool that embodies the proposed process in real world scenarios, a set of interfaces in a general framework embodiment, the validation and evaluation of the model with other platforms SDK, and a formal experiment to gather quantitative and qualitative data about the work produced using the adapted applications through our approach. The evaluation of the data in other collaborative scenarios, such as map navigation, music playing, video editing among others, can increase the knowledge of mobile development patterns, user interaction and attitudes in the context of collaborative work.

ACKNOWLEDGMENT

The authors wish to thanks the anonymous reviewers that made suggestions to significantly improve the presentation of this paper. This work has been funded by FAPESP/Brazil under grant No. 2012/04260-4 provided to the authors.

REFERENCES

[1] Agustina, A., Liu, F., Xia, S., Shen, H., Sun, C. CoMaya: Incorporating Advanced Collaboration Capabilities into 3D Digital Media Design Tools. In: Proceeding of the 8th ACM CSCW Conference. New York, USA, pp. 5-8 (2008)

[2] AndroidSamples, http://developer.android.com/resources/samples/

[3] Bartel, J., W., Dewan, P. Towards multi-domain collaborative toolkits. In CSCW '12 Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work, pp. 1297-1306 (2012)

[4] Begole, J., Smith, R., B., Struble, C., A., Shaffer, C., A. Resource sharing for replicated synchronous groupware. IEEE/ACM Transactions on Networking 9 (6), pp. 833-843 (2001)

[5] Cook, C., Churcher, N. Constructing real-time collaborative software engineering tools using CAISE, an architecture for supporting tool development. In: Proceedings of the 29th Australasian Computer Science Conference. Darlinghurst, Australia, pp. 267-276, (2006)

[6] Li, D., Li, R. Transparent sharing and interoperation of heterogeneous single-user applications. In: Proceedings of the 5th ACM CSCW Conference. New York, USA, pp. 246-255 (2002)

[7] Li, D., Yang. Y., Creel, J., Dworaczyk, B. A Framework for Building Collaboration Tools by Leveraging Industrial Components. Proceedings of the 2006 Confederated international conference on the Move to Meaningful Internet Systems. Montpellier, France, pp. 605-624 (2006)

[8] Lin, K., Chen, D., Dromey, R. G., Xia, S., Sun, C. API design recommendations for facilitating conversion of single-user applications into collaborative applications. In: Proceedings of the 3rd International Conference on Collaborative Computing: Networking, Applications and Worksharing. New York, USA, pp. 309-317 (2007)

[9] Lin, K., Chen, D., Sun, C., Dromey, R. G. Leveraging Single-User Microsoft Visio for Multi-user Real-Time Collaboration. In: Proceedings of the 4th International Conference of Cooperative Design, Visualization and Engineering. Shanghai, China, pp. 353-360, (2007)

[10] MathDoku, http://code.google.com/p/mathdoku/

[11] Morris, M. R., Cassanego, A., Paepcke, Winograd, T., Piper, A. M., Huang, A. Mediating Group Dynamics through Tabletop Interface Design. IEEE Computer Graphics and Applications Vol. 26(5), pp. 65-73 (2006)

[12] Pichiliani, M. C., Hirata, C. M. A Technical Comparison of the Existing Approaches to Support Collaboration in Non-Collaborative Applications. In: Proceedings of the 10th International Symposium on Collaborative Technologies and Systems. Maryland, USA, pp. 314-321 (2009)

[13] Roy, B., Graham, N., Gutwin, C. DiscoTech: a plug-in toolkit to improve handling of disconnection and reconnection in real-time groupware. In Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work, pp. 1287-1296 (2012)

[14] Schuckmann, C., Kirchner. L., Schümmer, J., Haake, J., M. Designing object-oriented synchronous groupware with COAST. In: Proceedings of the 3rd ACM CSCW Conference. New York, USA, pp. 30-38 (1996)

[15] Schuster, D., Springer, T., Schill, A. Service-based Development of Mobile Real-time Collaboration Applications for Social Networks. In: Proceedings of the 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), Mannheim, Germany, pp. 232-237 (2010)

[16] Springer, T., Schuster, D., Braun , I., Janeiro, J., Endler, M., Loureiro, A., A., F. A Fexible architecture for mobile collaboration services. Proceedings of the ACM/IFIP/USENIX Middleware '08 Conference. New York, USA, pp. 118-120 (2008)

[17] Sun, C., Xia, S., Sun, D., Chen, D., Shen, H., Cai W. Transparent adaptation of single-user applications for multi-user real-time collaboration. ACM Transactions on Computer-Human Interaction 13, pp. 531-582 (2006)

[18] Xia, S., Sun, D., Sun, C., Chen, D., Shen, H. Leveraging single-user applications for multi-user collaboration: the CoWord approach. In: Proceedings of 9th ACM CSCW Conference. Chicago, USA, pp. 162-171 (2004)

[19] Yang, Y., Li, D. Separating data and control: Support for adaptable consistency protocols in collaborative systems. In: Proceedings of the 10th ACM CSCW Conference. New York, USA, pp. 11–20 (2005)