WatchMyPhone - Providing Developer Support for Shared User Interface Objects in Collaborative Mobile Applications

Sven Bendel
Institute of Computer Science III
Rheinische Friedrich-Wilhelms-Universität Bonn
Bonn, Germany
bendel@cs.uni-bonn.de

Daniel Schuster
Computer Networks Group
Technische Universität Dresden
Dresden, Germany
daniel.schuster@tu-dresden.de

***Abstract*—Developing collaborative mobile applications is a tedious and error-prone task, as collaboration functionality often has do be developed from scratch. To ease this process, we propose WatchMyPhone, a developer toolkit being focused on the creation of collaborative applications on mobile devices, especially sharing user interface components like text views in a multi-user setting. We provide an implementation of the toolkit as well as a demo application for shared text editing based on Android. WatchMyPhone is integrated in the Mobilis open source framework thus adding another facet to enable fast and efficient development of mobile social apps.**

***Keywords*-CSCW, collaborative work, developer toolkit, mobile computing, awareness, shared editing**

I. Introduction

Many mobile applications today include social features like exchanging messages, location-awareness, content sharing, or access to remote objects on other user's devices [1]. Collaborative mobile applications take it to the next level and provide a more integrated and focused way of interacting with other users in real-time. Collaboration means working towards a common goal in a sort of shared environment. Building collaborative mobile applications still remains a challenge as we discovered during work on extending two existing location-based games (Mister X Mobile [2] and GeoQuest [3]) with collaborative features.

While the developer can build upon a lot of well-established concepts and algorithms in this field, software development support for collaborative mobile applications is rather weak. By this term we mean synchronizing two or more instances of the same mobile applications on different devices to be able to work on common artifacts. This functionality, which is often called replicated application sharing, can not be added transparently to single-user applications. It effects all layers of mobile applications like user interface (collaboration awareness), application logic (data synchronization), as well as network communication (collaborative network protocols). Thus, developer support should include all these layers which is not the case with existing collaboration frameworks.

To ease the development of collaborative mobile applications we present an approach to easily enhance single-user versions of user interface objects (widgets) to multi-user versions which can be used in almost the same way by the developer as their single-user predecessors. The state of these widgets is modeled as an XML data structure which is then synchronized among the participants of a collaboration session using XML-based Operational Transformation (OT). Support on the network layer is given by using XMPP-based presence and communication.

We implemented the proposed approach in a toolkit called *WatchMyPhone (WMP)*. This client-centered toolkit allows the app developer to focus on the development of core features, using *Developer Components*, which are part of the application GUI, and *Support Components*, which provide the data synchronization and network protocols needed for collaboration.

Section III explains the general architecture behind our solution. Section IV sketches the two WMP components. Finally, we present a prototype implementation in Section V which is used as a proof of concept.

II. Related Work

There is a number of existing systems or toolkits for mobile social applications like SAMOA [4], MobiSoc [5] and MobiClique [6] providing collaborative functionality on mobile devices. But these systems are mainly targeted at message exchange, media sharing, and social networking yet not offering the application sharing functionality as mentioned above. Our own work, the Mobilis framework [7], also does not support application sharing yet, but offers a service for shared editing of XML objects, which may make a good starting point for synchronizing application state.

Middleware and programming environments for mobile applications like M2MI [8], MoCA [9], or AmbientTalk [10], [11] offer means to access remote objects (or context tuples like in LIME [12]) on other devices in a network-aware way. Network-awareness means abstraction over networking technologies, reactions upon network (un)availability, and dynamic adaptation of network behavior (e.g., resize picture before transmission). This solves mobility problems on the network level and enables access to remote objects such as media files, but again offers no direct support for sharing applications, esp. synchronizing UIs.

Looking at the research area of application sharing, there are three approaches to what is called "retrofitting" collaborative features in existing applications [13]: at the operating system level, at the application level, and at the programming environment level. The first approach is what is known by many popular VNC implementations, like RealVNC [14] or TightVNC [15], i.e., sharing the actual output of an application using hooks inside the operating system. Besides VNC, collaborative video viewing has already been demonstrated on mobile devices [16] and could also be used for application sharing at the application level. While this works with almost every application from scratch, the data rate to be used is tremendous.

The second approach (application level) uses APIs of existing single-user applications (e.g., in CoWord [17] or CoMaya [18]). This is suitable if these APIs are already existing, offer comprehensive access to the applications, and are well-documented. As such APIs do not exist for most mobile apps, there is a big effort for the developer in creating and maintaining such APIs.

The third approach of retrofitting at the programming environment level seems most promising for mobile applications. Early attempts like Flexible JAMM [19] tried to add collaborative features transparently for the developer by replacing single-user UI components of desktop applications with their collaborative counterparts at runtime. As stated in the introduction, this often proves to be impossible due to the many effects of collaboration at different system levels. Later approaches thus tend to relax the transparency requirement by using Aspect Oriented Programming like in Zipper [13]. While AOP is a quite good solution for desktop applications, it is only poorly supported on mobile platforms. E.g., the popular AspectJ [20] framework generally can be used on the mobile OS Android, however, as Android doesn't allow bytecode modification at runtime it can only be applied to user-generated code. Thus it is not possible to weave code into system libraries like UI components.

Our approach also works at the programming environment level, but retrofits collaboration aspects by providing collaborative versions of system libraries on the mobile platform (esp. Android). In the next sections, we will outline why this approach fits best for mobile application development.

III. Architecture overview

Although the WMP toolkit is supposed to be used on the client side of an application, the general concept behind WMP also consists of a server component. Figure 1 gives an overview of the general architecture of WMP.

Client and server both operate in the same WMP network. Within the WMP network every user has a unique ID of the form `client@server`. This identification is used by all WMP services, i.e., for collaboration by WMP views (see below) and communication by chat. At every point in time every user additionally has a unique status, which can be *offline, online or busy*. The status defines which sorts of interactions are possible with this particular user.

The status is set automatically by the client application depending on whether he is logged into the network (which means he is *online*) or he is already taking part of a collaboration session (*busy*). The state can be retrieved from the communication server. If a user is *online*, he may enter an already existing collaboration session of a *busy* user. Both users have to use the same client application to do this - as WMP may be used by many applications this is not necessarily always the case. At every point in time a user may only take part at most at one session. Chatting is always available for all *online* and *busy* users.

The developer of a collaborative mobile app doesn't have to set up a server on his own. He may use a generic server which is shared by all applications backed by WMP in the beginning. Of course, with a growing user community, the setup of an own server is preferable. This should result in more developers starting to implement new collaborative mobile apps.

The server side consists of two independent entities: the *communication server* and the *collaboration server*. The communication server is solely responsible for communication among the clients and between clients and the collaboration server. In our impementation, we selected an XMPP server as communication server, but other (proprietary) communication servers may be used as well.

The collaboration server implements the server part of the collaborative editing framework being used for concurrent modifications on the same object (e.g., CEFX [21] or Google Wave [22]). In addition to this, the collaboration server also manages a map with a reference to the session a user is currently taking part of. This is used to determine the status of each user.

The WMP toolkit itself is solely used on the client. The next section will give an overview of the components of the toolkit and how they work together.

IV. WMP components

The WMP toolkit consists of two main components: the *Developer Components* and the *Support Components*.

A. Developer Components

The *Developer Components* are used by the developer while implementing his app. The most important part of the Developer Components is made of the *WMPViews*. WMPViews are collaborative versions of their stock OS view pendants. As an example, consider a simple text editing field (in Android this is called `EditText`). When a developer wants to have a collaborative text editor, he has to use the

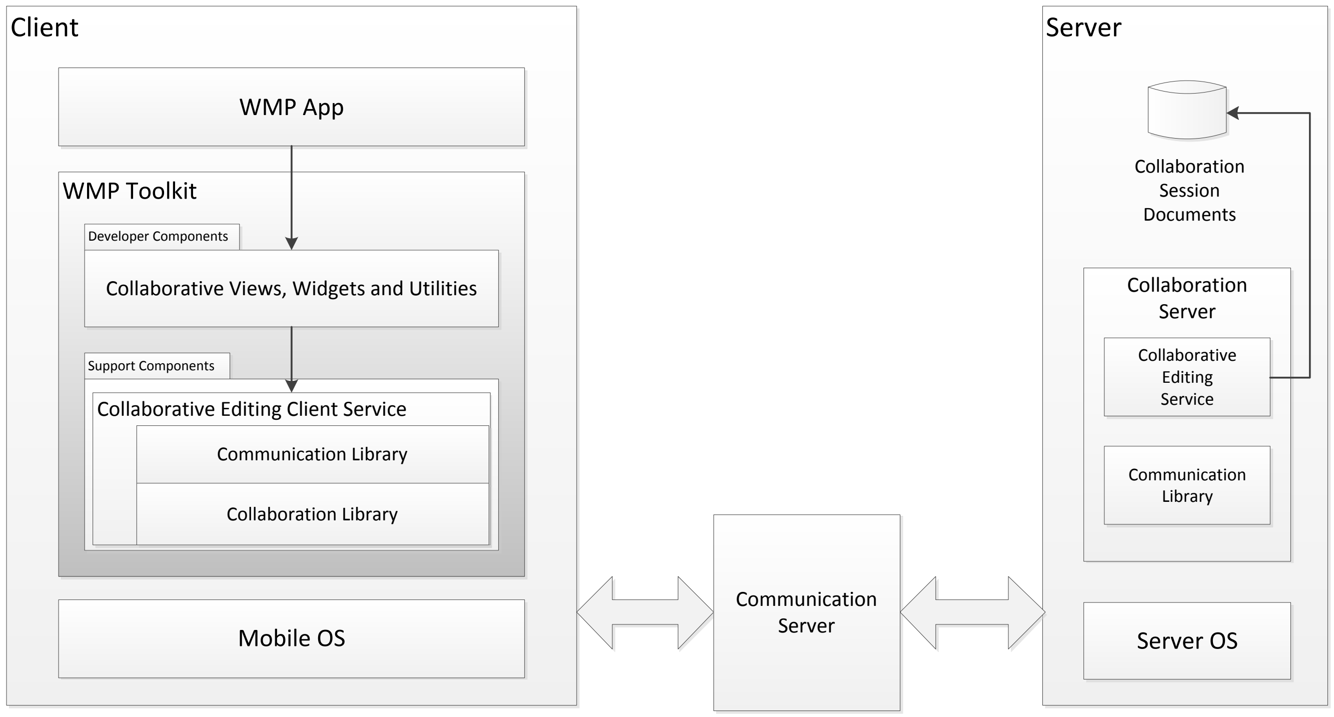


Figure 1. WMP architecture

`WMPEditText` view. This view will now automatically take care of all things needed for collaboration by making use of the *Support Components* assuming that we are already in a session (*MU-Mode*, short for *Multi-User-Mode*). But this doesn't mean, that this view doesn't work in *SU-Mode* (*Single-User-Mode*) as well. In fact, as we're subclassing the stock OS view, all features of these views are still available ensuring a consistent feature set in both modes. However, when in MU-Mode, the user expects more of the view. This is, where *Awareness Widgets* come in.

Awareness Widgets in general provide an enhanced user experience by unveiling the actions of other members of the collaboration sessions. WMP distinguishes between two sorts of Awareness Widgets: *internal* and *external* ones. Internal Awareness Widgets are directly built into their referencing WMPViews. Following the example from above this could be the highlighting of foreign text changes. External Awareness Widgets are independent GUI components. They obtain their information from their subject WMPViews which in turn are informed about changes by the Support Components.

A good example for an external Awareness Widget is the so-called *Radar View*. A Radar View displays the complete document in a condensed form and overlays it by the viewports of all participants. A user's viewport is simply the part of the document which is currently viewed by this specific user. Awareness Widgets ease the process of collaboration and help avoiding collisions when editing the same document. At this point it is important to emphasize, that not only text documents may be edited collaboratively, but also pictures, maps or even videos, provided that WMPViews and Awareness Widgets have been implemented for these tasks.

Moreover, the WMP concept allows for selective application sharing, i.e., sharing only part of the application state. This is especially useful in our mobile gaming use cases, where game state and locations of players should be synchronized among the participants but the rest of the UI elements are for individual use only.

In addition to WMPViews and Awareness Widgets, the Developer Components also consist of what we call *Collaboration Utilities*. Collaboration Utilities are not really that important for the proper functioning of collaboration sessions themselves, but rather provide support in the setup phase or during direct participant-to-participant communication. Examples for Collaboration Utilites are:

- *Login View*, which takes care of logging the user into the WMP network.
- *Buddy List*, showing the local user's friends together with their current status and allowing direct interaction such as initiating/taking part in chats or collaboration sessions.
- *Chat*, exchanging text messages between the session participants.

These utilities do not necessarily have to be used in WMP apps. One could think of scenarios, where ad-hoc session creation and one-time logins could be advantageous which would make Login View and Buddy List obsolete.

B. Support Components

In contrast to the Developer Components, *Support Components* are not directly used by the app developer, but by the Developer Components. As indicated above they provide the technical backbone for collaboration and communication matters. The core of the Support Components is the *Collaborative Editing Client Service*. This service provides the link between the libraries within the Support Components and the Developer Components and it holds the XML representation of the current session. This session XML consists of the XML representations of the content of all active collaborative views splitted up in nodes each corresponding to one collaborative view.

The two libraries are the *Communication Library* and the *Collaboration Library*. The first one handles all things related to communication between clients and between clients and server, providing a reliable way to communicate 1:1 and 1:many. We do not specify the communication libary in more detail, as it is exchangeable and not the main focus of our work. We decided to use the XMPP protocol for our implementation, but a mobile distributed middleware like the ones mentioned in Section II can also be used with our concept. In case of a P2P system, the communication server will disappear and there needs to be a mechanism to find a peer who takes over the role of the collaboration server.

The communication library also includes handling of network connectivity problems. It contains a message queue for phases of disconnection to be able to provide reliable data transfer. Nevertheless, in some cases it may be better to propagate this information to the app developer so he may react if he desires to.

The collaboration library mainly consists of the client-side implementation of a framework for shared editing of XML documents (including operational transformation), like Google Wave or CEFX. The idea behind this functional separation enables to easily change both libraries while leaving the Developer Components untouched. This provides a rather flexible solution.

C. Adding a new Developer Component

To give a hint about how to add new developer components, we are going to outline the theoretical development process for a collaboratively usable Canvas. The only prerequisite for a collaborative view is, that it's content can be represented by an XML structure which may be synchronized by the Collaborative Editing Client Service. So the component developer has to find an XML representation of the drawing, which he would probably also want to do, if he developed an ordinary single-user canvas.

Then he creates a subclass of an appropriate view from the system library which implements the `WMPView` interface. This interface allows the binding of the view to the Collaborative Editing Client Service, so that the view may be informed about changes and informs the respective views on other devices of it's own changes. The latter is done by calling various collaboration related methods (like `INSERT`, `UPDATE` and `DELETE`) at the service which modify this specific view's node in the session XML held by the service. So if the user draws a line the service is informed by the view of the `INSERT`ion of a new line which then takes care of adding this information to the canvas' node in the session XML (after having applied OT on it) and propagating the change to the other devices. On these devices the local Collaborative Editing Client Service also `INSERT`s the new line into the appropriate node in the XML session document and informs the canvas view about the change which then draws the new line.

V. Implementation and usage

We implemented a prototype implementation of the concept explained above using the Mobilis framework [7] and CEFX [21] for the mobile OS Android version 1.6. Additionally, a prototype application was written which shall serve us as a showcase of what is possible and how easy a developer may use our toolkit for the implementation of collaboratively usable mobile applications.

A. Implementation of WMP

The current version of the WMP toolkit makes use of the Mobilis framework to gain access to its very sleek and intuitive XMPP protocol implementation for Android. This is used for communication means between the clients as well as the clients and the server. An unmodified Openfire XMPP server [23] controls the communication. The Mobilis framework also already incorporates the CEFX framework for collaboration on XML documents.

The *Collaborative Editing Service* may be used by all WMP based applications on the device to handle collaboration. While the Mobilis Framework already provided a Collaborative Editing Service based on CEFX before we started work on WMP, we greatly overhauled and extended the service to meet our needs. The needed features of the aforementioned support libraries are provided by CEFX (collaboration/shared editing) and the "Mobilis XMPP for Android" implementation (communication).

It is important to mention that while our implementation is based on the Mobilis framework for simplicity reasons, it does not depend on it, as we already pointed out in Sections III and IV. Modularization of the implementation enables to exchange both, the communication service as well as the collaboration service, easily.

B. Integrating WMP in Android applications

If a developer wants to use WMP, he has to embed the WMP Android library. This library provides all necessary Developer Components, such as the `WMPEditText` being a subclass of the stock Android view for text editing, or the `RadarView` as a sample Awareness Widget. Their purpose is outlined in V-C. In case of the `WMPEditText` all user input (when in multi-user mode) is trapped and routed through the CEFX Operational Transformation framework. During this process the event is propagated to the other users using an XMPP multi-user chat [24], which was generated by the server.

The Collaborative Editing Service holds a local XML document which mirrors the state of the shared view, by assigning a node to every `WMPView`. For the `WMPEditText` the content of the node would be an XML representation of the text field's content. An update of this document by the local or a remote user triggers an event which is processed by the Collaborative Editing Service which eventually updates the associated `WMPView`, **after** the XML session document has been updated. This helps maintaining a consistent state. When in single-user mode this obviously doesn't apply, as there is no session document maintained. Thus in single-user mode the application just works as if WMP had not been there.

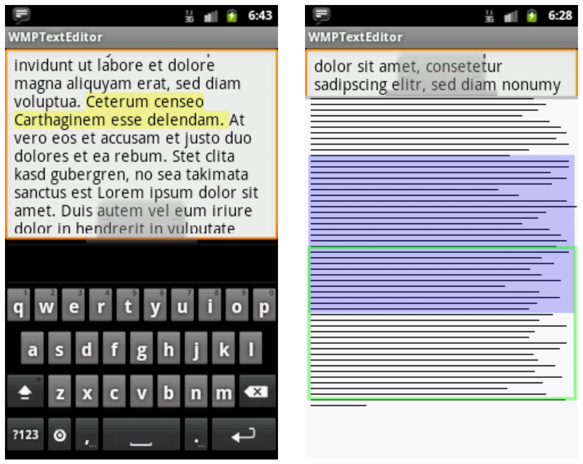
As explained above, external Awareness Widgets are tied to a subject view which is always a `WMPView`. This view informs its `WMPAwarenessWidgets` about changes. Based on the specific type of change the Awareness Widget decides, whether it reacts on it or not.

To Android developers it will be of particular interest, that in case of the Android OS, the binding process between Awareness Widget and `WMPView` can be directly initiated from the XML file describing the Activity layout. This can be done by referencing the collaborative view's ID from the Awareness Widget definition via the `app:subject` attribute (see Listing 1, line 11). The figure also shows how easily collaborative Views and widgets may be integrated into an application when using WMP on Android. In many cases most of the integration effort is done by just declaring them in the XML file like any other Android View. They will work out of the box without any additional Java code, although there is always the possibility to (un-)bind them from there, too.

```
 1  <edu.bonn.cs.wmp.views.WMPEditText
 2     android:id="@+id/main_edit_text_collab"
 3     android:layout_width=" fill_parent "
 4     android:layout_height =" fill_parent "
 5  </edu.bonn.cs.wmp.views.WMPEditText>
 6
 7  <edu.bonn.cs.wmp.awarenesswidgets.RadarView
 8     android:id="@+id/main_radar_view"
 9     android:layout_width=" fill_parent "
10     android:layout_heigth =" fill_parent "
11     app:subject="@+id/main_edit_text_collab">
12  </edu.bonn.cs.wmp.awarenesswidgets.RadarView>
```

Listing 1. Binding an Awareness Widget to a WMPView with WMP on Android

In contrast, internal Awareness "Widgets" are directly maintained within the `WMPViews` code and need no binding.



invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. Ceterum censeo Carthaginem esse delendam. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Duis autem vel eum iriure dolor in hendrerit in vulputate

dolor sit amet, consetetur sadipscing elitr, sed diam nonumy

(a) Collaborative text field (b) Radar View

Figure 2. Prototype screenshots

C. Demo application

We implemented a basic collaborative text editor which acts as a proof-of-concept. The editor makes use of all up to now implemented Developer Components of WMP. These are a collaboratively editable text field with integrated awareness widget (highlights text changed by remote users) and a hideable Radar View to display the viewports of all users. As it relies completely on WMP technology it can be used in multi-user as well as in single-user mode without loosing any functionality - even the Radar View is fully functional. For screenshots see Figure 2. The implementation is straight-forward and contains just a few lines of Java and XML code, with both mainly handling the UI.

Running the application on a Google Nexus S with Android 2.3.7 and a HTC Desire with Android 2.2.1 we found the performance to be rather good. On input of arbitrary words response time (= time between input and output on the display) most of the time stayed below the mark of 100 ms. Performance over longer time periods and with longer documents remain to be tested.

VI. Conclusions

We have outlined WatchMyPhone as a developer toolkit which supports mobile application developers in integrating collaborative components into their applications. This is done via a set of views, Awareness Widgets and utilities (Developer Components) backed by a service utilizing a collaboration and a Communication Library (Support Components). The developer uses only the Developer Components while the Support Components take the responsibility for handling communication and collaboration issues which means that the developer doesn't have to care about these aspects. This enables broader usage of collaborative technologies in mobile applications.

The current implementation is just for Android devices and is based on the Mobilis framework. A standalone version (also for other mobile operating systems) would be desirable.

There are also just a handful of Developer Components at the moment and no utilities implemented. We would like to extend this set in the future.

Nevertheless, WMP is just the first step towards providing easy-to-use general-purpose collaboration for mobile apps. It would be interesting to see a UI-centered approach like WMP be combined with network-aware mobile middleware better dealing with network connectivity problems, providing ad-hoc connectivity, and adaptation to changing network conditions. The question of how much of these issues should be handled by the app developer remains an open research question.

References

[1] T. Springer, D. Schuster, I. Braun, J. Janeiro, M. Endler, and A. Loureiro, "A flexible architecture for mobile collaboration services," in *ACM/IFIP/USENIX Middleware'08 Conference Companion*, Leuven, Belgium, 2008.

[2] Qeevee UG, "Mister X Mobile," http://qeevee.com/projects/misterx, 2011.

[3] Software Architecture and Middleware, University of Bonn, "GeoQuest," http://sam.iai.uni-bonn.de/projects/geoquest/index.html, 2012.

[4] D. Botazzi, R. Montanari, and A. Toninelli, "Context-Aware Middleware for Anytime, Anywhere Social Networks," *IEEE Intelligent Systems*, vol. 22, no. 5, pp. 23–32, 2007.

[5] A. Gupta, A. Kalra, D. Boston, and C. Borcea, "MobiSoC: a middleware for mobile social computing applications," *Mobile Networks and Applications*, vol. 14, no. 1, pp. 35–52, 2009.

[6] A.-K. Pietiläinen, E. Oliver, J. LeBrun, G. Varghese, and C. Diot, "MobiClique: middleware for mobile social networking," in *2nd ACM workshop on Online social networks (WOSN'09)*, Barcelona, Spain, 2009.

[7] R. Lübke, D. Schuster, and A. Schill, "A Framework for the Development of Mobile Social Software on Android," in *The 3rd International Conference on Mobile Computing, Applications, and Services (MobiCASE)*, Los Angeles, CA, USA, 2011.

[8] A. Kaminsky and H. Bischof, "Many-to-many invocation: a new object oriented paradigm for ad hoc collaborative systems," in *Companion of the 17th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (OOPSLA)*, Seattle, WA, USA, 2002.

[9] V. Sacramento, M. Endler, H. Rubinsztejn, L. Lima, K. Goncalves, F. Nascimento, and G. Bueno, "Moca: A middleware for developing collaborative applications for mobile users," *Distributed Systems Online, IEEE*, vol. 5, no. 10, pp. 2–2, 2004.

[10] T. Van Cutsem, S. Mostinckx, E. Boix, J. Dedecker, and W. De Meuter, "Ambienttalk: Object-oriented event-driven programming in mobile ad hoc networks," in *XXVI International Conference of the Chilean Society of Computer Science (SCCC '07)*, Iquique, Chile, 2007.

[11] K. Pinte, D. Harnie, and T. D'Hondt, "Enabling cross-technology mobile applications with network-aware references," in *13th International Conference on Coordination Models and Languages (COORDINATION)*, Reykjavik, Iceland, 2011.

[12] A. Murphy, G. Picco, and G. Roman, "LIME: A coordination model and middleware supporting mobility of hosts and agents," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 15, no. 3, pp. 279–328, 2006.

[13] L.-T. Cheng, J. Patterson, S. L. Rohall, S. Hupfer, and S. Ross, "Weaving a Social Fabric into Existing Software," in *4th International conference on Aspect-oriented software development (AOSD '05)*, Chicago, USA, 2005.

[14] RealVNC Limited, "RealVNC - VNC remote control software," http://www.realvnc.com/index.html, 2012.

[15] TightVNC Group, "TightVNC: VNC-Compatible Free Remote Control / Remote Desktop Software," http://www.tightvnc.com/, 2012.

[16] G. Shen, Y. Li, and Y. Zhang, "MobiUS: enable together-viewing video experience across two mobile devices," in *5th international conference on Mobile systems, applications and services (MobiSys '07)*, San Juan, Puerto Rico, 2007.

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

[18] A. Agustina, F. Liu, S. Xia, H. Shen, and C. Sun, "CoMaya: incorporating advanced collaboration capabilities into 3D digital media design tools," in *ACM conference on Computer supported cooperative work (CSCW)*, San Diego, CA, USA, 2008.

[19] J. Begole, R. B. Smith, C. A. Struble, and C. A. Shaffer, "Resource sharing for replicated synchronous groupware," *IEEE/ACM Trans. Netw.*, vol. 9, no. 6, pp. 833–843, 2001.

[20] The Eclipse Foundation, "The AspectJ Project," http://www.eclipse.org/aspectj/, 2012.

[21] A. R. S. Gerlicher, "Transparent Extension of Single-User Applications to Multi-User Real-Time Collaborative Systems - An Aspect Oriented Approach to Framework Integration," in *Ninth International Conference on Enterprise Information Systems (ICEIS)*, Funchal, Madeira, Portugal, 2007.

[22] D. Wang, A. Mah, and S. Lassen, "Google Wave Operational Transformation - Google Wave Federation Protocol," http://www.waveprotocol.org/whitepapers/operational-transform, 2010.

[23] Jive Software, "Ignite Realtime: Openfire Server," http://www.igniterealtime.org/projects/openfire/, 2011.

[24] P. Saint-Andre, "XEP-0045: Multi-User Chat," http://xmpp.org/extensions/xep-0045.html, 2008.