TouchSync: Lightweight Synchronization for Ad-Hoc Mobile Collaboration

Vassil Roussev, Gabriel Perez Priego, Golden G. Richard III
Department of Computer Science
University of New Orleans
New Orleans, LA 70148
vassil@roussev.net, golden@cs.uno.edu

Abstract

For current generation of PDA and smart phone devices, wireless capabilities are practically a standard feature. Consequently, they should be able to support users in their spontaneous daily interactions. Yet, the original host-centric model of data synchronization is still entrenched, thereby depriving users of adequate support for ad-hoc collaboration. In our view, what is needed is a more relaxed and intuitive sharing model that reflects the semi-structured nature of informal user interactions.

We present TouchSync—a lightweight framework that allows users to easily form an ad-hoc group and automatically share data on their devices. The framework provides a flexible sharing mechanism that allows various single-user applications to be easily adapted for collaboration. Adapted applications completely retain their original user interaction functions, while the infrastructure provides generic collaboration tools for session management and privacy control. We discuss both the overall design of the framework and an implementation for PalmOS and Bluetooth. We also present implementation results from the adaptation of three common applications to support meetings.

KEYWORDS: Ad-hoc Collaboration, Lightweight Synchronization, CSCW, Mobile Collaboration

1. INTRODUCTION

The original success of *Palm* PDA devices was due to their novel ability to share data with a host workstation via a serial cable. This eventually enabled the development of mobile applications that allowed users to collaborate asynchronously by sharing data via the network capabilities of the host workstation. Current generation PDAs (and smart phones) feature built-in wireless network capabilities that enable devices to talk to each other directly, potentially supporting real-time spontaneous collaboration. While individual applications with such capabilities have been built, there has been little work in providing system support that would make the creation of such applications routine and would allow users to manage them in generic way. Furthermore, current work does not properly address important privacy issues that inevitably arise from data sharing.

To understand the requirements for ad-hoc mobile collaboration, let us consider a spontaneous meeting. In this scenario, all participants come with their *Bluetooth*-enabled PDAs and would like, at a minimum, to be able to share sketches and text notes taken during the meeting and be able to schedule their next meeting. Even in this simple case, we can identify several important **usability requirements**:

- *Standardized applications*. The applications used in the collaboration should be familiar to everyone. Ideally, all common tools, such as *Memo* and *Calendar,* should be supported.
- *Standardized collaboration management*. The process of establishing and controlling a collaborative environment should be application-independent.
- *Flexible privacy policy*. Users should be in complete control of the sharing process and should clearly understand what is being shared and with whom.
- *Robustness*. The system should gracefully handle problems such as participants walking in and out of range.
- *Ease of use*. Although implicit in most systems, this requirement is particularly important here because ad-hoc collaborators would not want to invest much time in learning special skills and will be more inclined to abandon the system at the first sign of trouble.

Practically all of these requirements point to the need for a (software) infrastructure-supported solution

that would provide the basis for standardizing the collaboration. From the point of view of the developer, the most important infrastructure requirement is

- *Automation* (ease of implemention). In particular, it should be easy for the developer of a single-user application to transform it into a collaborative one using the infrastructure. In other words, the development effort using the infrastructure should be *much smaller* than without it

In the following section, we briefly discuss the most relevant existing work and the extent to which it supports the above requirements.

2. RELATED WORK

Ad-hoc collaboration has been an active topic of research for some time and various approaches have been proposed. This section uses representative systems to summarize different ideas relevant to our work and is not an effort to exhaustively enumerate all related systems.

IBM's Instant Collaboration project [2] focuses on providing a smooth transition from informal (collaboration-related) interaction to semi-structured "activity-centered" collaboration, to formal collaboration where group goals and activities are well-defined. The essential idea is to present the notion of shared objects and to provide awareness of the actions of group members (e.g., user *A* is working with the object now). Users are given an *ActivityExplorer* [6], which provides them with a familiar tree-based interface that in turn leads users to other detail windows (chats, comments, notes, etc.).

Instant Collaboration focuses primarily on collaborations in a classic work environment where people work on their desktops and, generally, belong to the various branches of the same organization. Mobility is not an explicit concern and the entire system relies on a relatively heavyweight middleware infrastructure that needs to be deployed beforehand.

Proem [8], [9] is a relatively early work on proximity-based peer-to-peer networking. It follows a classic toolkit-based approach, where applications are developed using a common toolkit. In its aim to be generic, however, it provides only relatively low-level communication abstractions that leave quite a bit of work to the application developer.

JXTA (http://jxta.org) is representative of a number of distributed infrastructure approaches that have the ambition of solving all issues surrounding ubiquitous computing by means of imposing a unifying standard. As with other related efforts, the problem is that such infrastructures must invariably provide lower level abstractions than the ones used by applications. In the case of JXTA, the basic model is that processes exchange messages via unidirectional unreliable pipes [7]. In essence, JXTA allows the building of a *logical* peer-to-peer overlay network by providing basic naming and service discovery services, and by abstracting away the underlying communication network. Thus, higher-level infrastructure components are needed to provide the level of abstraction needed for direct use by application developers.

One representative system is *Expeerience* [1], which is a middleware layer over JXTA. Its main focus is on addressing issues with JXTA with regard to intermittent connections in ad-hoc environments. It supports *mobile services*, which are based on *Java* code mobility in a manner that is very similar to *Jini* (http://jini.org). While code mobility is an attractive and extremely flexible approach, there is an inherent security risk associated with it, especially in (typically untrusted) ad-hoc peer-to-peer environments. Therefore, it is unlikely that this kind of approach would be widely acceptable.

P2Pcomp [2] attempts to get around the relatively heavyweight JXTA approach by sticking to a very minimalist component model that could potential work with a number of available communication mechanisms, including JXTA. That avoids the need for code mobility, but imposes a prerequisite infrastructure deployment.

Speakeasy [3] uses *recombinant computing* [2] as a more structured take on the code mobility approach. It combines design patterns with code migration to achieve "serendipitous interoperability". All *Speakeasy* components implement one or more of a small number of interfaces that fall into four categories: *data transfer*, *collection*, *metadata*, and *control*. The general expectation is that end-users will have to make the ultimate decisions at run-time on how the different components fit together.

The problem, however, is that this implies that there is no easy, standardized way to control the collaboration, as the controls will be combined on the fly and would likely vary from application to application. In our view, this is a non-trivial problem and, along with the

already mentioned trust issues, is a major impediment in moving recombinant computing from the lab to the real world.

Sync [8] is a framework for asynchronous collaboration targeted primarily at mobile device although there are no fundamental obstacles to applying them in other scenarios. Like most similar systems, *Sync* keeps track of object changes during periods of disconnect and later merges the device version of the client (seen/edited by the user) with a server version of the object (indirectly seen/edited by other users). The most notable feature of the system is its ability to flexibly define the outcome of conflicting updates via *merge tables*. A merge table entry specifies what should happen if the version of a data entity (field, record, table entry) has changed on the client, on the server, or both. The possible outcomes include, accepting the client/server version of it, recursively invoking the merge procedure at a finer granularity, or asking the user to interactively resolve the conflict.

This table-driven approach allows for a lot of flexibility yet it is a simple abstraction that can be used with any application and can be presented in a uniform way to the user. Furthermore, default policies can be specified at various levels of granularity: user (covering all applications used by that user), application (covering all instances of the application), or individual objects. Hence, the specification effort is proportional to the level of customization required by the user.

In summary, existing approaches to ad-hoc mobile collaboration generally rely on two basic approaches to solve the problem of interoperating devices. The first one is the deployment of a unified middleware infrastructure, which enables all infrastructure-enabled devices to work together. The second one is based on mobility where initial infrastructure requirements are minimal and missing functionality can be dynamically obtained by downloading code. In general terms, the former approach is less flexible but does not have the security issues of the latter. Some systems employ a hybrid approach that provides more predictability with respect to dynamically acquired code.

3. SYSTEM DESIGN

After careful review of existing work, we decided to take a different approach from previous work. First, we decided against code mobility due to its inherent security problems. Next, we opted for a minimalist approach which attempts to solve our specific problem rather than develop yet another expansive distributed infrastructure that tries to be everything to everyone. Ideally, we would like to create a small module that could be integrated with the operating system. Thus, we would solve the problem of deploying the infrastructure components without undue complexity. In many respects, this work is an effort to show by example that system support can be provided at a very modest cost.

3.1 Rationale

The entire *Palm* interaction model is based on the idea that the user is using a GUI to edit database records one at a time. As soon as the editing of a record is completed, the update is committed and the new status of the record is noted to aid the subsequent synchronization. This works reasonably well as the record database (RDB) is typically placed on a flash memory card and the (relatively small) update can performed fast enough not to cripple responsiveness.

From a software engineering point of view, this can be viewed three-layer model: user interface (GUI), core application, and persistence (RDB). The sharing (exchange of data) functions could be implemented at any one of them. However, some layers are better candidates than others—the core application layer obviously varies widely as a function of application semantics, whereas the GUI (widgets) and persistence layers are provided by the OS and are shared among all applications. Thus, providing a sharing solution at the GUI or RDB layers would provide more automation than application-level sharing.

The problems with sharing the GUI (i.e., presenting all users with the same UI) are several: screen/input capabilities vary, all participants must have the exact same application installed, connectivity must be maintained at all times, and every user input event must be shared. In order to maintain consistency, the resulting system has to limit concurrency by allowing input from one user at a time.

Based on the above analysis, our approach is to implement the collaborative sharing functions at the RDB layer. It has none of the GUI-sharing drawbacks: different *Palm* devices could interoperate, users could use their favorite implementation in a class of applications (such as a calendar), and users can interact concurrently with the applications. In fact, they could be interacting with any application on their device

(shared or not) and still receive the shared state. Figure 1 illustrates the point: User 1's updates are shared among all participant, whereas Users 2 and 3 interact only with their local databases. For all practical purposes, those two users could be running any application while the updates take place.

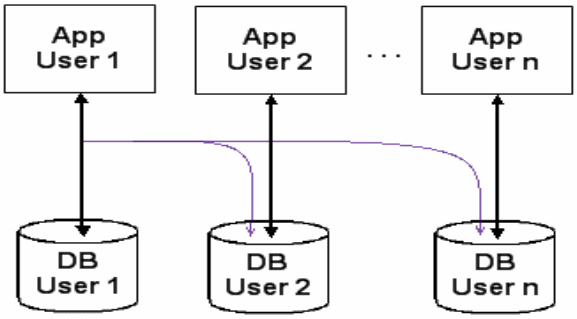

Figure 1 Example *TouchSync* interaction

Finally, unlike on desktop systems where hard drive latency limits collaboration to asynchronous mode, persistence layer sharing on devices can be near real-time because updates are small, incremental, and committed frequently.

3.2 Conceptual Overview

The essential abstraction we present to the user is the shared record—whenever one of the users creates, modifies, or deletes a record, the rest of the collaborators will automatically receive the new version. For example, if somebody in the group makes a new note and saves it, it will automatically be propagated to the group (subject to privacy settings). Similarly, if somebody modifies the note, all will transparently receive the new version, which will overwrite the old one.

From a user's point of view, automated sharing raises obvious privacy concerns. Therefore, we give the user complete control regarding what data is shared, with whom, and how it is shared. For every application, a specific policy can be given from a set of predefined ones, or a custom one can be specified on the fly, as shown in the next section.

To the best of our knowledge, existing systems either do not address privacy management or have stuck to the classic access control mechanisms, such as ACLs. In our view, none of these are adequate because spontaneous collaboration often means interacting with strangers. In that case doing nothing (leaving the system open) is obviously not a wise option, as demonstrated (in the extreme) by the Bluetooth "rifle" experiment [11] in which cell-phone data was collected from a large distance from unsuspecting users.

Using traditional "heavyweight" access control mechanisms is often not easy as these are typically tied to a specific administrative domain. Creating special users and user roles for incidental use requires too much effort and is rarely justified. Furthermore, keeping track of access rights over the long term and having an intuitive understanding of which remote users may have access to local information is often an undue burden on the user.

Based on these observations, we set out to design a privacy management mechanism that is application-neutral, lightweight, and intuitive. Recall that the basic scenario we want to support is a meeting of a small group of people with PDA-like devices. The participants may belong to different organizations, they may be meeting for the first time, and do not necessarily trust each other with anything else beyond the information exchanged during the meeting.

The first design decision we made was to share only records modified during the current collaborative session. In other words, we do not attempt to 'synchronize' the complete record databases of different users but only to facilitate the sharing of the records related to the current session (meeting). This gives users a readily identifiable time reference that helps them intuitively understand what is shared. For example, passive users who have not edited a record are assured that none of their information is sent out. For the active users, we provide a policy-based control mechanism that allows them to decide who can receive their changes.

Table 1 Sharing specification example

Remote Action	Local record status			
	Not existent	No change	Updated	Deleted
Insert	insert	---	---	---
Update	insert	update	local	insert
Delete	no action	delete	insert	no action

The other side of the privacy problem is the concern that a newly received record might overwrite an old record and, for example, mess up a participant's schedule. To address this, we give users the option to specify how received updates should be handled based on the sender and the status of the local record. We distinguish among three types of record changes:

insert, *delete*, and *update*. The sharing mechanism makes a decision based on a table specified by the user and is similar in spirit to *Sync*'s merge policy specifications [10]. For example, Table 1 specifies a policy that favors the actions of the local user over those of remote ones with the exception of local deletes, which are overridden by remote updates.

In the next section, we briefly describe the main features of our implementation and the control interface seen by the users.

4. IMPLEMENTATION

TouchSync consists of three functional components: a session manager, a tool for ad-hoc sharing for databases (ASDB), and a privacy manager. The session and privacy managers are completely application-independent, whereas applications need some adaptation to work with ASDB.

4.1 Communication

All communication is based on the PalmOS-provided Exchange Manager, which enables the sending and receiving of typed data objects, such as MIME data, databases, or database records. Applications use the Exchange Manager to send and receive typed data objects. The Exchange Manager interface is independent of the transport mechanism. Applications can use IR, SMS, Bluetooth or any other protocol that has an Exchange Manager plug-in called an exchange library (Figure 2).

The Exchange Manager works in conjunction with an exchange library. Each exchange library is transport-dependent and performs the actual communication with the remote device. When an application makes an Exchange Manager call, the Exchange Manager forwards the request to the appropriate exchange library. The Exchange Manager's main duty is to maintain a registry of libraries implement protocols for various types of data.

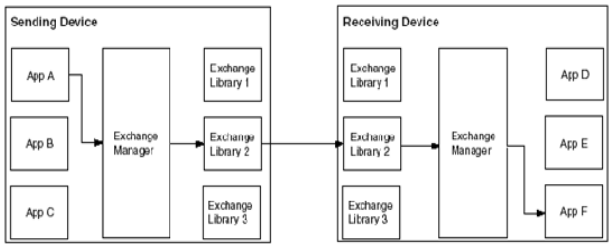

Figure 2 Palm Exchange Manager

4.2 Session Management

The first step in any collaboration is to establish a group session. In our system, this initiated by one of the participants who becomes the host of the session and may serve as a moderator.

As shown on Figure 3*, after the host starts the ASDB tool and clicks on the `<Start Session>` button, a *Bluetooth* discovery is initiated and the host selects the participants from a list of available devices. Once the participants are selected, the host application creates a piconet, stores the addresses of the members, and sends the list (including himself) of participants to the rest of the members. After the initial handshake with all participants, the wireless connection is closed (to save the battery).

It is worth noting that, after a session is established, it is not necessary for the host to be present and collaborators are free to walk in/out of the session and no additional setup is needed. Synchronization is handled automatically as users rejoin and more users can be added to the session as needed. In the case of latecomers, one caveat is that, in order for them to receive preceding changes, the relevant records need to be touched. While it is quite straightforward to handle this automatically, we chose not to in order to preserve the basic premise of privacy control—do not transmit records that have not been explicitly modified. This is analogous to spoken conversation in a real meeting—if latecomers need to be brought up to date, it must be done explicitly.

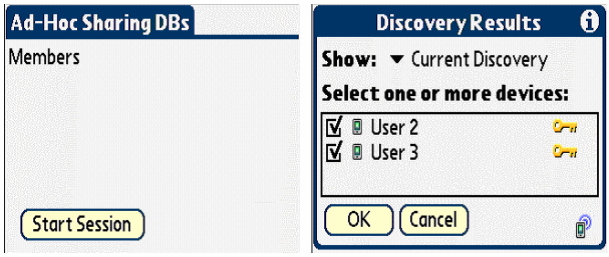

Figure 3 Host: Initiating a session and selecting participants

At this point, users switch to the shared version of the application they want to use, which behaves exactly as the single-user one but also performs automatic updates of remote RDBs. Ideally, we would like to have a sharing process that is completely transparent to

* All screenshots have been cropped/resized to fit the paper format.

the application. Unfortunately, current versions of the *PalmOS* do not appear to provide a suitable framework for making this a reality. In particular, there is no event mechanism that would allow the sharing to be automatically triggered by a commit to the database. Therefore, we found it unavoidable to ask for the help of the developer in adapting the applications.

4.3 Application Adaptation

Figure 4 illustrates the idea of the adaptation process. The original application's functions are extended to allow interoperation with the ASDB tool, which implements the actual sharing.

The necessary changes to the application are relatively small and are restricted to the functions dealing with the record database. Namely, the application needs to add to its regular database update methods a routine notification to the ASDB. To detect the user changes, we modify the saving and deleting record methods of the *Palm* application. This modification includes creating a structure in which we store the record and its application information.

After initializing the structure we pass it to the ASDB application which is in charge of sending the record to the other members in the session. For that purpose, we use the launch application option provided by Palm—applications can send launch codes to each other, so an application might be launched from another application. An application can use a launch code to request that another application perform an action or modify its data. In our case sending sending a launch code to another application on a remote device works much like a remote invocation. Thus, an additional method is needed to process incoming remote updates and store them in the local database.

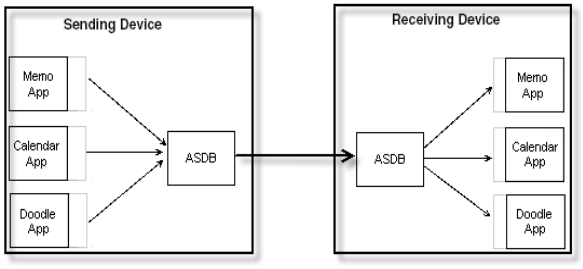

Figure 4 Application adaptation in *TouchSync*

The latter is invoked by the ASDB using *Palm*'s `SysAppLaunch` method. This turns out to be especially convenient since remote updates can be applied to a database even if the user has not launched the application. In fact, the user could be using an altogether different application at the time the update is received.

For our proof of concept implementation, we chose three common applications that would benefit any meeting—Memo, Calendar, and Doodle (a sketch note editor). We tested our framework on three different Palm OS handhelds: a SONY Clié PEG-NZ90, a Palm Tungsten T, and a Palm Tungsten T3. The exact hardware/software configurations are shown on Table 2

Table 2 Test device configurations

	Resolution	OS	Processor	Memory
Tungsten T	320x320	5.0	144MHz	16MB
Tungsten T3	320x480	5.2.1	400MHz	64MB
Sony Clié	320x480	5.0	200MHz	11MB

The point of testing on different devices was simply to verify that our code does, indeed, interoperate across different *Bluetooth* implementations and different Palm OS versions. The main focus of our evaluation, however, was to quantify and qualify the effort involved in converting a regular Palm application into a collaborative one. Table 3 and Table 4 provide a numeric summary of our findings.

Table 3 Adaptation effort

	Calendar	Memo	Doodle
Lines of code (original app)	28,890	6,481	2,492
Lines of code added	370	150	90
Percentage of adaptation	1.28%	2.31%	3.61%

Table 4 Complexity of adaptation effort

	Calendar	Memo	Doodle	Total
Methods Added	3	3	2	8
Methods Modified	3	3	5	11
C Files Added	1	1	0	2
C Files Modified	2	1	1	4
Headers Added	1	1	1	3
Headers Modified	0	0	0	0
Structures Added	3	3	3	9

As the numbers demonstrate, the lines of code necessary are between 1 and 4% of the size of the original application and were concentrated in 6-7 methods. We estimate that, for a developer familiar with the original applications, the effort for all three would comfortably fit in 2-3 days.

Since the core application functions remain the same, users would not notice any difference in the application behavior they are used to. The only

additions are the notifications produced by the *Palm* Exchange Manager, such as the one on Figure 5.

The `<Skip>` button allows the sender to bypass a particular user for the current send. Similar notifications appear on the recipients' displays to keep users aware of the data exchanges taking place.

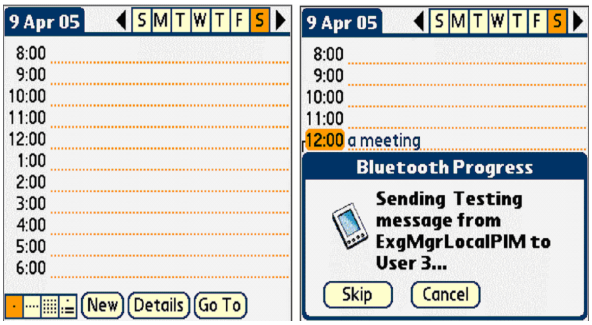

Figure 5 *Bluetooth* message exchange notification

4.4 Privacy Management

A privacy policy is a set of rules through which users controls the sharing for each application. *TouchSync* has a policy manager that stores the rules and applies them each time a record is sent/received. Generally, a policy specifies two things: who should receive updates from the local user and how should remote updates be installed locally (Figure 6).

Given the improvised collaboration scenarios we are trying to support, we have opted for a simplified, Unix-style access control scheme with respect to different principals. In particular, only the host can be singled out for special handling with the assumption that the host also serves as the social moderator of the meeting.

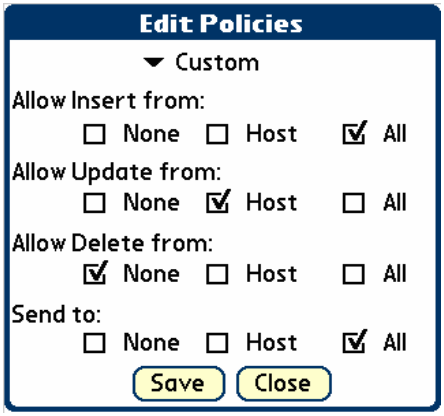

Figure 6 Privacy policy specification: Definition

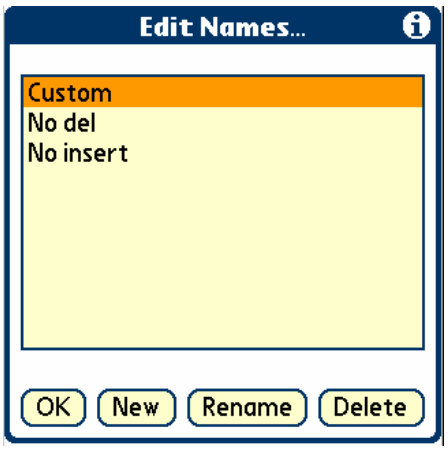

Figure 7 Privacy policy specification: Naming

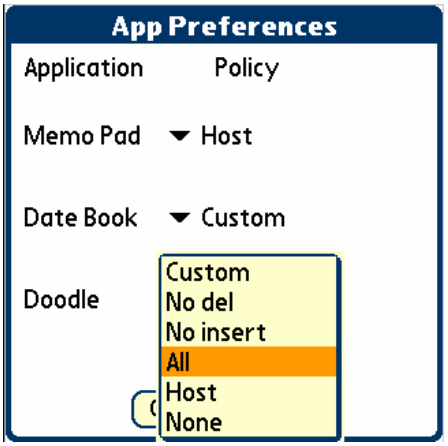

Figure 8 Privacy policy specification: Application binding

Once a policy is defined, it can be named so that it can be reused later (Figure 7). Finally, for each application, a policy binding is established through the application preferences tab (Figure 8). Once a binding is set, it becomes effective immediately and will persist across sessions until a new one is given. If no user policy is supplied, a default one is used.

5. CONCLUSIONS

In this paper, we presented initial results of our work on *TouchSync*, a lightweight framework for ad-hoc collaboration on *Palm* devices. Its main benefits include:

- *Support for commonly used applications*. The current implementation provides shared versions of three common meeting tools—*Memo*, *Calendar*, and *Doodle* (sketch editor).

- *Standardized collaboration management*. *TouchSync* presents a simple and intuitive interface for establishing and managing collaborative sessions that is independent of the set of shared applications. The actual data sharing is based on synchronizing updates to the participants' record databases and is handled automatically by the system.
- *Flexible privacy policy*. The user has full fine-grain control of the sharing process and can define the desirable outcome of conflicts between local and remote updates through a simple table interface.
- *Robustness*. The system is resilient to drops in communication and allows dynamic group membership. Retransmitted data is correctly identified and is recorded as an in-place update rather than a new (redundant) record.
- *Easy to use*. *TouchSync* retains the original single-user interaction of the shared applications. The additional collaboration-related behavior is controlled through a uniform, application-independent interface, which consists of only 5 new screens (all shown here).
- *Automation*. Due to operating system restrictions, it is not possible to provide a fully automated solution without hacking the kernel. Short of that, *TouchSync* provides a highly automated solution. In our experiments, the coding effort was between 1 and 4% of the original application and was concentrated in 6-7 methods.

6. FUTURE WORK

There are several directions in which we would like to expand our work on *TouchSync*:

- Gain more experience by adapting more complex applications and develop a library that would bring the effort of implementing a new application to zero.
- Perform a field study to observe actual usage of the system.
- Add more advanced features, such as logging/replay and multi-user undo/redo
- Cross-platform interoperability—include standards-based support (e.g., *SyncML*) for other PDA and mobile phone platforms.

REFERENCES

[1] Bisignano, M., A. Calvagna, G. Di Modica, and O. Tomarchio: "Expeerience: a JXTA middleware for mobile ad-hoc networks", Proceedings of the Third International Conference on Peer-to-Peer Computing (P2P), 2003.

[2] Edwards, W., Newman, M., Sedivy, J., Smith, T., and Izadi, S. "Challenge: Recombinant Computing and the Speakeasy Approach". In Proceedings of the Eighth ACM International Conference on Mobile Computing and Networking (MobiCom 2002). Atlanta, GA. Sep, 2002.

[3] Edwards, W., Newman, M., Sedivy, J., Smith, T., Balfanz, D., Smetters, D., Wong, H., Izadi, S., Shahram Izadi. "Using Speakeasy for Ad Hoc Peer-to-Peer Collaboration". In Proceedings of the ACM 2002 Conference on Computer Supported Cooperative Work (CSCW), New Orleans, LA, Nov 2002.

[4] Ferscha, A., Hechinger, M., Mayrhofer, R., Oberhauser, R., "A Light-Weight Component Model for Peer-to-Peer Applications". In Proceedings of the 24th International Conference on Distributed Computing Systems Workshops - W4: MDC (ICDCSW'04)

[5] Geyer, W., Cheng, L., "Facilitating Emerging Collaboration through Light-weight Information Sharing," In Conference Supplement: ACM 2002 Conference on Computer Supported Cooperative Work (CSCW), New Orleans, LA, Nov 2002.

[6] Geyer, W., Vogel, J., Cheng, L., Muller, M, "Supporting Activity-Centric Collaboration through Peer-to-Peer Shared Objects," Proceedinds of the Group 2003 Conference (GROUP), Sanibel Island, FL, Nov 2003.

[7] JXTA v2.3.x: Java™ Programmer's Guide. http://www.jxta.org/white_papers.html

[8] Kortuem, G. "Proem: A peer-to-peer computing platform for mobile ad-hoc networks." In Advanced Topic Workshop Middleware for Mobile Computing, Heidelberg, Nov 2001.

[9] Kortuem, G., Schneider, J., et al. "When peer-to-peer comes face-to-face: Collaborative peer-to-peer computing in mobile ad-hoc networks." In First Intl. Conf. on Peer-to-Peer Computing (P2P), pages 75–91, Sweden, Aug 2001.

[10] Munson, J., Dewan, P., "Sync: a Java framework for mobile collaborative applications", IEEE Computer. 1997. p. 231-242.

[11] Zetter, K., "Security Cavities Ail Bluetooth", Wired.com, Aug 8, 2004.