5-20-2005

Ad-Hoc Sharing for Palm Devices

Juan Gabriel Perez Priego
University of New Orleans

AD-HOC SHARING FOR PALM DEVICES

Thesis

Submitted to the Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirements for the degree of

Master of Science
in
Computer Science

by

Juan Gabriel Pérez Priego

B.S, University of the Americas – Puebla, México, 2000

May 2005

Dedication

To my mother’s memory.

You will always be in my heart.

Acknowledgements

I want to express my gratitude to my advisor Dr. Vassil Roussev for his constant support, guidance and encouragement.

In addition, I would like to thank Dr. Shengru Tu and Dr. Nauman Chaudhry for being on my thesis defense committee.

I want to thank my brothers, especially Chely, for all the support and care they provided to me.

I would like to especially thank Carmen for her priceless love, support, and encouragement. My life will not be the same without her. Cosha, thanks for being with me.

The most special acknowledgment to my parents, Jose Luis Perez and Rosy Priego who have encouraged and guided me in my life.

Table of Contents

List of Figures

List of Tables

Abstract

The current generation of Palm PDA devices is designed to share information records primarily with a base desktop system, or a server. Therefore, their built-in features for sharing data during ad-hoc collaboration among groups of mobile users are inadequate.

In this thesis, we describe a new framework that addresses this problem by allowing users to transparently share the record databases of common applications during spontaneous collaborative sessions. The framework also allows users to define custom sharing policies for each application/user pair. These policies determine the manner in which records are exchanged and update, thereby automating the process of handling conflicts and preserving user privacy preferences.

We also present implementation results, in which we have used the framework to create shared versions of common applications, such as Calendar and Memo. Our experimental results show that the programming effort involved is minimal and the user interaction with the application is, essentially, the same as in the original application.

Chapter 1: Introduction

Personal Digital Assistants (PDA) are one of the fastest growing areas of computer technology. They are now more accessible and their features have increased as well. PDAs are becoming an important part of working life for many people. In a PDA, users can store documents, addresses, appointments, videos, pictures, etc. People use their PDA or similar devices for professional, educational, commercial or entertainment activities.

1.1 Palm OS™ Devices History

Alan Kay, a graduate student at the University of Utah in the 1970s, first described the PDA. His idea was to have an interactive computer similar to a book with wireless communications abilities and a flat panel display. The device he described was called the Dynabook **[1**].

In 1996 Palm Inc. launched the Palm Pilot 1000 and 5000. These Palms were designed with 128 and 512 KB of RAM respectively; they included basic applications like memo, calendar, address book, calculator, and some games. During the next years Palm present several models (Palm III, Palm IIIx, Palm V) in which some features such as memory, processor speed, and battery type, were improved. In 1999 Palm Inc. was the first company to release a PDA with wireless Internet access, Palm VII, using its own Internet Service Provider Palm.Net [1].

Figure 1.1 Palm Pilot with 128KB of RAM.

The Palm IIIc model in 2000 was the first Palm device with color display. In 2001, Palm presented the M500 which included an expansion slot compatible with Secure Digital (SD) and Multi Media Card (MMC). These hardware allow users to increase the storage [2].

Current Palm models (Tungsten T5) offer and increasing range of functionality such as E-Mail, and multimedia capability, word processing, spreadsheet and data base software, internet access and Bluetooth communication [3].

Figure 1.2 Palm Tungsten T|5 with 256MB of internal flash memory.

1.2 Motivation

Due to the personal nature of PDA, most of the applications today are developed for a single user. However, the PDA users work in collaborative environments in which the user needs to exchange information or share ideas with others. Even though current Palm models include Bluetooth communication, there are few Palm applications that take advantage of this for providing collaboration.

To make our discussion more specific consider the following scenario of a team of workers and see how the Palms have influenced their activities. We start with the simple case in a work meeting where technology is not used. In this meeting, some people take notes in a notebook, share a board where any person can write down his/her ideas and schedule the next meeting in their calendar. Once the PDA appeared on the scene even though the PDA was not very common, the members' behavior changed somewhat. A user could bring his Palm to the meeting, and he takes notes in his Palm, keeps for his own or sends later to others using his PC. Scheduling the next meeting among several group members could be very complicated. With his Palm the user could easily check his calendar and state when he could be available and schedule the next meeting on his Palm, but the other users probably would have to wait to check theirs on their PCs or common calendars.

After the Palm became more available, more users could bring their own Palm to work meetings. Each user takes their personal notes and at the end of the meeting they can share their notes and schedule the next meeting. However if some users did not have enough time, they could not share their information with the others.

The process of sharing the information in a meeting will be more efficient if it starts at the beginning of the meeting and keeps working throughout the meeting. This will be a more

efficient use of meeting time and will assure that all users have the same information at the end of the meeting.

The goal of this work is to allow ad-hoc group of users to transparently share their records of common applications during spontaneous collaborative sessions. The requirements that we have considered for the design of a collaborative handheld application are:

1. Establishing an ad-hoc session.
2. Sharing their records in a natural way.
3. Allow to the user to customize the level of sharing/privacy for each application.

The first step of our collaboration must begin when users with their Palms come together in the same place. One of them will host a session and choose the members that are going to collaborate.

Once the users are in the same session they will be able to share their information with others. When a user adds or modifies one of his records, the record is going to be sent to the other members without any intervention of the sender.

Besides letting the user share his information we must offer him a way to customize his level of sharing by creating different policies for each application/user pair. A policy is a rule which will define how the user's records are going to be exchanged and updated. This will assure the privacy preferences of the user's records.

Integrating these parts will give the Palm's user a tool to collaborate with others in a wide range of scenarios.

1.3 Thesis Organization

The rest of this thesis is organized as follows: Chapter 2 reviews the current state-of-the-art of collaborative PDA applications. Chapter 3 presents the design of our specific approach. Chapter 4 explains in detail the implementation of this framework. Chapter 5 includes our conclusions and the ideas being developed for future work.

Chapter 2: Related Work

It is useful to classify collaborative handheld applications in two categories according to how they share the information:

- Client-server
- Peer-to-peer

The first category refers to those applications that rely on a central server in which all the information is stored. The second category shows the applications whose communication happens directly between Palms.

We analyze the advantages and disadvantages of the applications based on the design requirements mentioned in the previous chapter.

2.1 Client-Server

The core of this type of system is that each device has to be connected to a network infrastructure. The devices send the information to the server and the information is available for anyone who has access to the server. Some server-based systems provide the functionality that when the devices are offline the server keeps all the changes made from other devices and when those offline devices become available the server will send them the changes.

This type of system works well because most of the processing relay in the server which reduces the work in the mobile devices. However, these systems depend on a network infrastructure which is not available everywhere. If users of the server based system have a meeting outside of their office it will be difficult for the users to collaborate among themselves and other users.

Client-server systems will not let users to have a free-mobility since they rely on a network infrastructure.

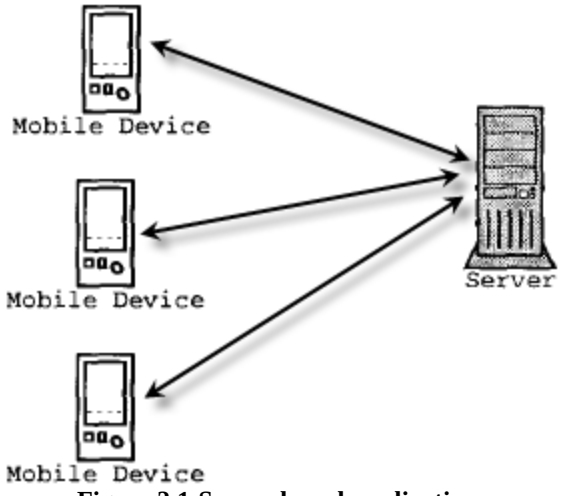

Figure 2.1 Server based application.

2.1.1 NotePals

"NotePals is an ink-based, collaborative note taking application that runs on personal digital assistants (PDAs). The system is distinguished by its support for lightweight collaboration at three levels: hardware, note taking process, and user interface." [4]

NotePals allows users to take their own sketches and notes. For the sketches users can easily draw them directly on the screen. For text notes, NotePals provides a focus area in which

users can write their text; the text is scaled down by a factor of 2 ½, this is copied in the main page that has a virtual resolution of 400 x 273. The NotePals design allows users to fit more text on a page, and it keeps the users' hand out of the way while writing.

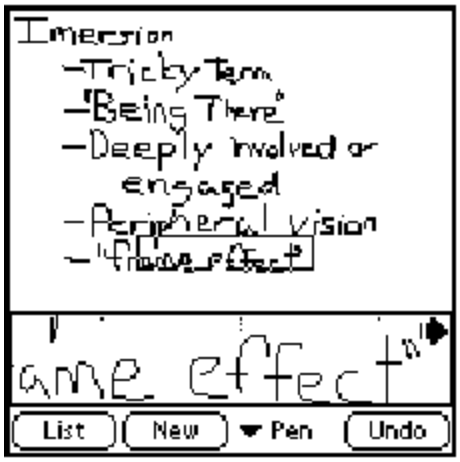

Figure 2.2 NotePals' user interface with focus area [4].

When a user synchronizes his Palm with his desktop computer, the notes that have been taken with NotePals are uploaded to a Web-based repository server which is responsible for storing and sorting all notes from all the members. Using a web browser, users can filter the notes by project, author, date, and type.

NotePals has the features for sharing information among the members of a group. However, the kind of information being shared is limited to a one type of data, "sketch" notes; the other limitation in NotePals is that the information cannot be shared on real-time, users have to wait until all the taken notes are uploaded in the server.

2.1.2 COMAL

COMAL (Collaborative Mobile Application Library) is a framework to develop handheld collaborative applications. This framework has a couple of libraries which provide a communication schema which can be used by developers allowing them to concentrate more on building the collaborative application itself and not in low level programming. COMAL is friendly with existing collaborative applications, it allows code sharing between different handheld applications, and provides different modes of exchange of information [8].

The framework libraries include:

- COMAL desktop library
- Palm handheld library

The COMAL desktop library has the function to send, receive and synchronize data to and from Palm devices. This library provides the mechanisms to receive requests for information from the Palm device, process the transaction and save the result for the next time the Palm is available. The library can also receive information having the handheld device as its final destination and, again, store it until delivery is possible.

The Palm handheld library is primarily made of wrappers around standard system calls of the Palm OS libraries. This library allows the exchange of data such as user lists, messages and events, objects commonly used in typical collaborative systems.

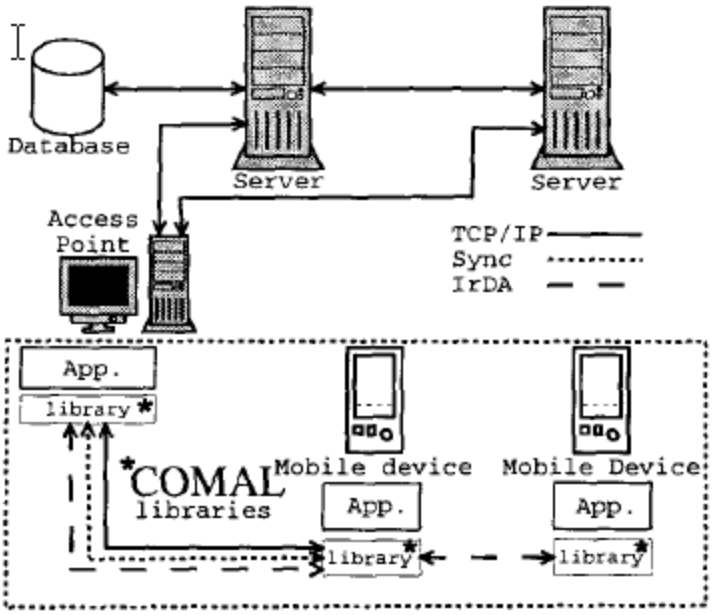

Figure 2.3 Role of the COMAL Library in a typical handheld collaboration [8].

Group Messenger is an application developed using COMAL libraries. The main purpose of this application is to allow a group of persons working in a collaborative environment to communicate among themselves. User information is stored on a server. The information can be accessed from different desktop computers, and handhelds connected to the network infrastructure. The exchange of data between the server and desktop clients is performed using TCP/DP socket connections. On the handheld side, information between desktop computers and the Palm device flows using the COMAL libraries.

COMAL provides an alternative way to develop collaborative applications with a few code lines of code to program. However, COMAL relies on a complex network infrastructure, which does not allow users to have a spontaneous collaboration in other places where there is not such infrastructure.

2.1.3 SyncML

SyncML is the open standard that drives data mobility by establishing a common language for communications between devices, applications and networks. The foundation of the SyncML open standard, SyncML Data Sync (SyncML DS) ensures a consistent set of data that is always available on any device or application, any time [9].

The SyncML was founded in February 2000 and it is supported by Ericsson, IBM-Lotus, Matsushita (Panasonic), Motorola/Starfish, Nokia, Openwave and Symbian.

SyncML enables synchronization to be performed in a standardized way across applications, devices and networks—whether it is synchronizing e-mail in a Palm handheld with that of a stand-alone PC via a Bluetooth or cable connection, or calendar entries on a mobile phone with those of the corporate network via the public mobile phone network [10].

SyncML uses the extensible markup language (XML) for specifying the messages that synchronize devices and applications (using either plain text or the wireless binary XML, WBXML, binary encoding technique employed by WAP).

SyncML is designed to be independent of data and transport type: it uses existing open standards for object types—such as vCal (calendar, to-do lists) and vCard (contacts) personal data formats—and can support arbitrary networked data.

SyncML can handle one- and two-way synchronization as well as client-and server initiated synchronization. To ensure full interoperability, a range of transport protocols is supported. These include the WAP wireless session protocol (WSP), hypertext transfer protocol (HTTP) and OBEX (local connectivity using Bluetooth, IrDA and RS-232). SyncML can also be deployed over networks that use SMTP, POP3, IMAP, pure TCP/IP, and proprietary wireless communication protocols.

The main components of the SyncML specification are:

- An XML-based representation specification.
- A synchronization protocol.
- Transport bindings for the synchronization protocol.

SyncML contains a set of well-defined messages that are conveyed between a client and server participating in a data synchronization operation. The messages are represented as XML documents or as multipurpose Internet mail extension (MIME) entities.

The representation specification specifies an XML document type description (DTD), which allows the representation of all information required to perform synchronization, including data, metadata and commands.

There are two roles in SyncML:

- SyncML client. Typically, this is a PC, mobile phone, or personal digital assistant (PDA). It contains a synchronization client agent and usually sends the SyncML messages (operations), possibly including payload data. It must also be able to receive responses from the SyncML server.
- SyncML server. Typically, this is a networked server or a PC. It contains a synchronization server agent and synchronization engine, and usually receives the SyncML messages (operations), possibly including payload data from the SyncML client.

The SyncML client pushes SyncML request messages to the SyncML server. The server synchronizes the data within the SyncML messages with data stored in the server (including additions, updates, and deletions). The SyncML server then returns a response to the SyncML messages sent by the client. The synchronization protocol supports one-way and two-way synchronization (client-to-server or server-to-client).

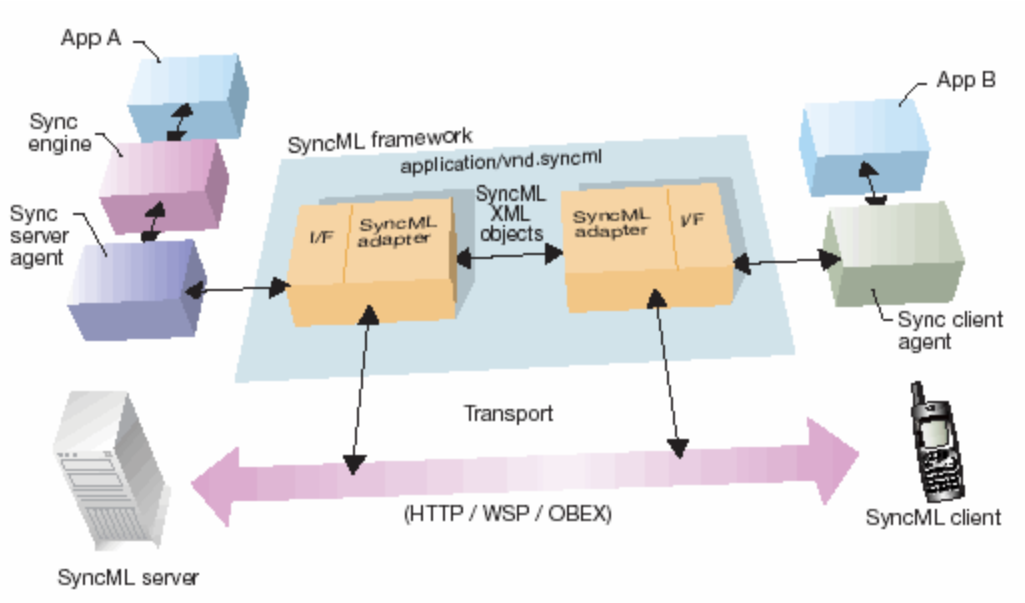

Figure 2.4 SyncML framework [10].

SyncML also identifies a small set of data formats that provide a set of media types for exchanging common accepted information, such as contacts, calendars, and messages. Support for these data formats is mandatory for conformance to the specification. In addition to these formats, SyncML allows for the identification of any other registered format. SyncML uses the MIME content type framework for identifying data formats, called MIME media types.

SyncML supports capabilities exchange that is, the SyncML client and server can determine which device, user, and application features the other device supports.

2.1.4 Summary

Client-server applications work well due to most of the processing side is made on the server which reduces the work in the mobile devices. These applications provide the collaboration when the users get the up to date information from the server. However, these applications do not work well for ad-hoc groups. The users' applications not only rely on a network infrastructure but also they depend on a configuration in both the applications and the server. This make difficult to integrate a new user into an existing group. These applications have been designed for working in predefined groups.

2.2 Peer-to-Peer

Peer to peer applications do not depend on a network infrastructure. The communication is established directly between mobile devices. This gives the users free mobility and they can work in any place. For example if a user has a first time meeting outside his office with other users, the users can share a P2P application and start a collaborative session among them without the need of a complex network infrastructure.

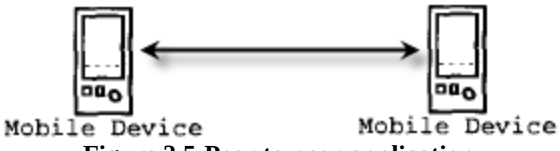

Figure 2.5 Peer to peer application.

2.2.1 Palm OS Beam

Palm OS provides a simple way for sharing information by using infrared communication. With infrared "beaming", users can send a business card, memo, or even a software program to another handheld user [11].

Most of the Palm applications include the option for using the infrared "beaming". When a user wants to share a record he selects the record and clicks beam option. The receiver has to be in the line of sight of the sender which in practical terms is a major impediment in ad-hoc collaboration. This is due to the user has to point and send to each user device.

Once the record is received the receiver has the option to accept or reject the record.

2.2.2 Palm OS Bluetooth Send

Bluetooth is a wireless protocol that is used to communicate from one device to another in a small area usually less than 30 feet. In Bluetooth the line of sight is not required. The process for sharing information is similar to that of the infrared beaming. When the sending process is started the sender device looks for another Bluetooth device in the range. The sender can choose one or more devices for sending his record.

Each time a user wants to share a record using Bluetooth, he needs to repeat all the process. In the scenario, when the user is in a meeting, repeating the process for sharing several records would be an inefficient use of meeting time.

2.2.3 Summary

Palm OS provides infrared and Bluetooth options for sharing information in a simple way. However, we cannot consider these as efficient ways to collaborate because the users have to follow the process every time they want to share information. Another limitation is that the records are duplicated each time they are received

The peer to peer applications are more suitable for ad-hoc collaboration. They are made for providing free mobility to users. However, all the processes are executed on the mobile devices. This can overcharge the devices which most of the time have poor resources (battery, memory, and processor).

Chapter 3: Design

Handheld devices have become a tool of common use due to the fact that they are small devices with helpful applications. In PDAs people have found the need for developing their work through collaborative sessions. As presented in the previous chapter, there are some projects that allow collaboration among PDAs. However, most of the solutions are client-server based. The limitation of this type of system is that they rely on a network infrastructure which limits the users' work area.

In our framework we give an approach for spontaneous collaboration. Our design is based on a peer to peer architecture. This design allows users to have free mobility and their devices do not require a central server for communication. This approach works well in a scenario where users have to work with new users in different places besides their main office.

Our framework allows an ad-hoc group of users to transparently share their records of common applications during spontaneous collaborative sessions. The characteristics that we consider in our design are:

- Session management. Allow users to establish an ad-hoc session, which defines who is going to participate in the sharing process.
- Record-sharing implementation. Allow users to transparently share their application records.

- Sharing customization. Allow users to customize the level of sharing/privacy for each application.

Our design includes two components. The first component is a Palm application called "Ad-hoc Sharing DB" (ASDB) that is in charge of establishing the communication among Palms. The second component is an adapted common Palm applications that allow users to collaborate among them.

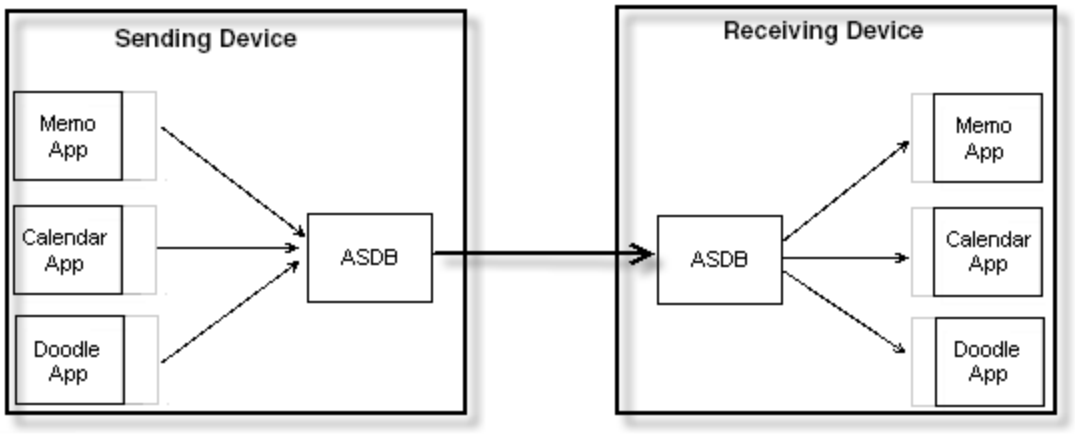

Figure 3.1 Diagram used in ASDB.

The ASDB application controls all the processes involved during the communication among the devices. The first process is called "handshake" which happens when the users meet in the same place, they start a session and define the members of the session. The second process is the exchange of records. This process will be executed as many times as the users work with the records of the adapted applications.

The ASDB application is also in charge of storing the users' policy preferences. These policies determine the manner in which records are exchanged and updated, thereby automating the process of handling conflicts and preserving user privacy preferences.

We adapt some common Palm applications (Calendar, Memo, and Doodle) in order to allow users to collaborate. We choose these applications because putting them together give users a complete tool for collaboration in meetings. Another reason for selecting these applications is that their records have different levels of complexity which give us a wide variety of application to include in our framework. The Memo record is a plain text. The Calendar record has a structure that includes different types of data such as date, time, alarm information, etc. The Doodle record is an image matrix.

The code used for modifying the applications is minimal. The modifications are:

- In the processes of the insert, update, or delete the record, we add the functionality that makes the application notify and pass the record to the ASDB application.
- In the process of receiving the record from other users, we insert the code that stores the record received through ASDB

The users' interaction with the adapted applications is, essentially, the same as in the original applications.

This design allows users to transparently share the record databases of common applications during spontaneous collaborative sessions. With the use of our framework and programming a few lines of code, we are able to provide a complete set of tools that can be used for collaboration. Our design is intended to be used for other Palm developers who are interested in adding collaboration functionality to their applications.

In our design we follow some of the features applied in a Virtual File System (VFS) Interface. A VFS is an abstraction of a physical file system implementation. It provides a consistent interface to multiple file systems, both local and remote [21]. In our framework during

the sharing process, we can seamlessly move among the applications' databases in different Palm devices (Figure 3.3). The sharing process open, read, write, and close the databases when the record is send from one user to the others. In our VFS approach our sharing process identifies the local databases from the remote ones.

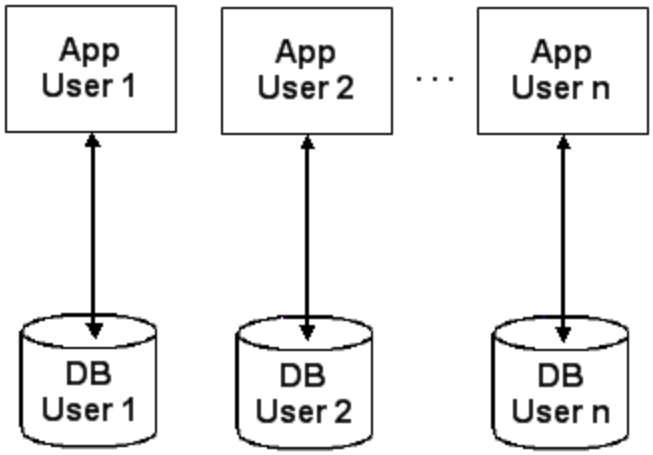

Figure 3.2 Normal Palm application approach

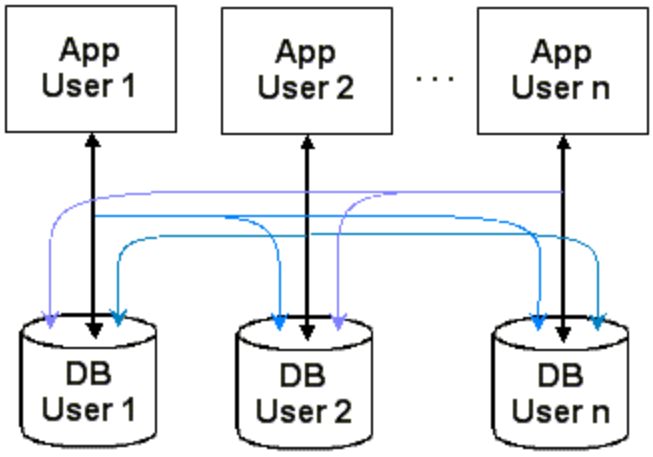

Figure 3.3 VFS approach.

Chapter 4: Implementation

In this chapter, we describe the implementation of our framework that addresses the problem of sharing records in a spontaneous collaboration among mobile users. Our framework also allows users to define custom sharing policies for each application/user pair. These policies determine the manner in which records are exchanged and updated.

4.1 Technology Used

We implemented our framework for Palm OS devices because they provide a well-constructed design for PDAs. Besides, Palm Inc. gives a good support for developers. Bluetooth has been implemented as a communication mean for many devices including the newest Palm OS handhelds. Palm OS includes a library that is used for exchanging DB records and applications among devices.

4.1.1 Palm OS DBs

In Palm OS applications the information is stored in databases. These databases can contain either records or applications. In Palm OS, databases organize related records; every record belongs to one and only one database. A database may be a collection of all addresses book entries, all datebook entries, and so on. A Palm OS application can create, delete, open, and close databases as necessary, just as a traditional file system can create, delete, open, and close a

traditional file. There is no restriction on where the records for a particular database reside as long as they all reside on the same memory card. The records from one database can be interspersed with the records from one or more other databases in memory [14].

A traditional file system first reads all or a portion of a file into a memory buffer from disk, using and/or updating the information in the memory buffer, and then writes the updated memory buffer back to disk. Because Palm OS devices have limited amounts of dynamic RAM and use nonvolatile RAM instead of disk storage, a traditional file system is not optimal for storing and retrieving Palm OS user data.

Palm OS accesses and updates all information in place. This works well because it reduces dynamic memory requirements and eliminates the overhead of transferring the data to and from another memory buffer involved in a file system. Databases can be distributed over multiple discontiguous areas of physical RAM due to the fact records can be stored anywhere on the memory card.

In Palm OS a database maintains a list of all records that belong to it by storing the local ID of each record in the database header. An application can access any record in a database by index. A database header consists of some basic database information and a list of records in the database. Each record entry in the header has the local ID of the record, 8 attribute bits, and a 3-byte unique ID for the record.

The database header has the following fields:

- Name holds the name of the database.
- Attributes has flags for the database.
- Version holds an application-specific version number for that database.

- ModificationNumber is incremented every time a record in the database is deleted, added, or modified.
- AppInfoID is an optional field that an application can use to store application-specific information about the database.
- SortInfoID is another optional field an application can use for storing the local ID of a sort table for the database.
- Type and Creator are each 4 bytes and hold the database type and creator.
- NumRecords holds the number of record entries stored in the database header itself.

Each record entry has the local ID of the record, 8 attribute bits, and a 3-byte unique ID for the record:

- Local IDs make the database slot-independent. Since all records for a database reside on the same memory card as the header, the handle of any record in the database can be quickly calculated.
- The unique ID must be unique for each record within a database. It remains the same for a particular record no matter how many times the record is modified.
- The delete bit is set in the attributes flags, but its entry in the database header remains until the next synchronization with the PC.
- The dirty bit is set whenever a record is updated.
- The busy bit is set when an application currently has a record locked for reading or writing.

- The secret bit is set for records that should not be displayed before the user password has been entered on the handheld.

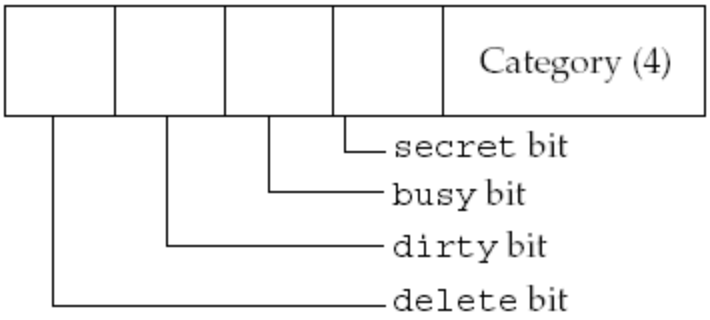

Figure 4.1 Record attributes [14].

4.1.2 Bluetooth

In 1998, five major companies (Ericsson, Nokia, IBM, Toshiba, and Intel) formed a group to create a license-free technology for universal wireless connectivity in the handheld market. The result is Bluetooth, a technology named after a 10th-century king who brought warring Viking tribes under a common rule. The Bluetooth specifications, currently in version 1.1, define a radio frequency (RF) wireless communication interface and the associated set of communication protocols and usage profiles [16].

The link speed, communication range, and transmit power level for Bluetooth were chosen to support low-cost, power-efficient, single-chip implementations of the current technology. In fact, Bluetooth is the first attempt at making a single-chip radio that can operate in the 2.4-GHz ISM (industrial, scientific, and medical) RF band.

The Bluetooth 1.1 specification was released in February 2001. The specification consists of two parts:

- Core Specifications: The core specification defines all layers of the Bluetooth protocol stack (Figure 4.2), the Bluetooth stack differs from the classical seven-layer networking model in some ways. These differences are primarily to support ad-hoc connectivity among participating nodes, while conserving power and accommodating devices that lack resources to support all layers of the classical networking stack.
- Profile Specifications: Vendors can use the services offered by the Bluetooth stack to create a variety of applications. Because interoperability is crucial to Bluetooth's operation, the Bluetooth SIG has defined profile specifications to support it. The current specifications include 13 profiles. The profiles specify controller and stack parameter settings as well as the features and procedures required for interworking among Bluetooth devices. All vendor implementations of these profiles are expected to be interoperable. The Bluetooth certification authority uses the profiles to test and certify compliance, and grants use of the Bluetooth logo only to products that conform to the methods and procedures defined in the profiles.

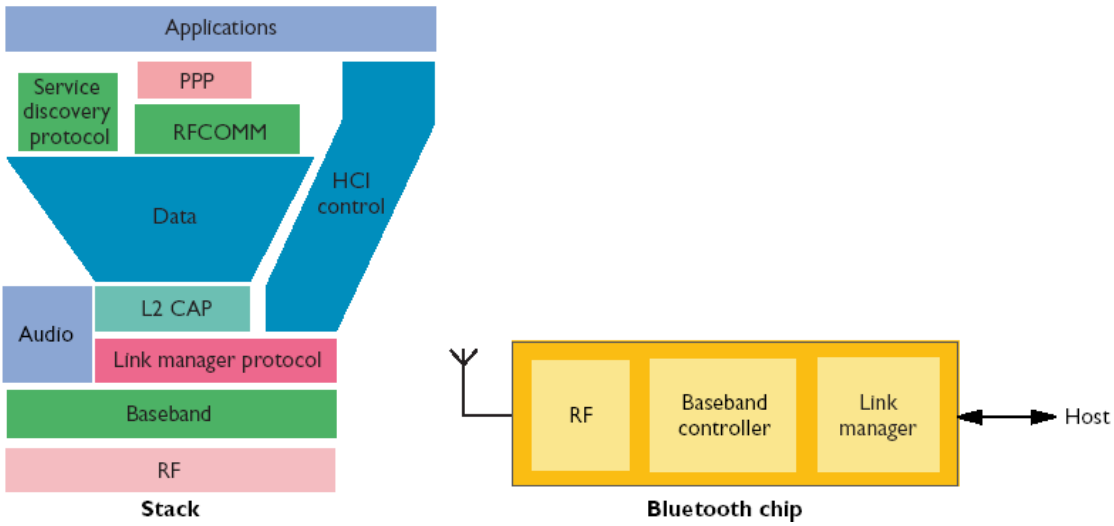

Figure 4.2 The Bluetooth networking stack and chip [17].

Bluetooth is a frequency-hopping spread-spectrum system. This means that the radio hops through the full spectrum of 79 or 23 RF channels using a pseudorandom hopping sequence. The hopping rate of 1,600 hops per second provides good immunity against other sources of interference in the 2.4-GHz band. The link speed is 1 Mbps, which is easily achieved using a simple modulation technique (Gaussian Frequency Shift Keying, or GFSK). A more complex modulation technique could achieve a higher rate, but GFSK keeps the radio design simple and low cost.

A set of Bluetooth devices sharing a common channel is called a piconet. As shown on Figure 4.4, a piconet is a star-shaped configuration in which the device at the center performs the role of master and all other devices operate as slaves. Up to seven slaves can be active and served simultaneously by the master.

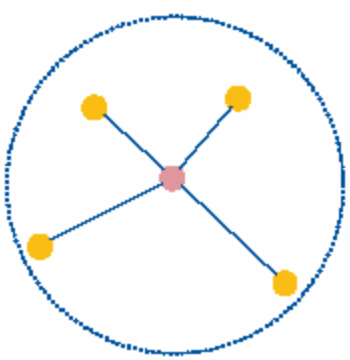

Figure 4.3 A Bluetooth piconet [16].

In some usage scenarios, however, devices in different piconets may need to communicate with each other. Bluetooth defines a structure called scatternet to facilitate interpiconet communication. A scatternet is formed by interconnecting multiple piconets. As shown Figure 4.5, the connections are formed by bridge nodes, which are members of two or more piconets. A bridge node participates in each member piconet on a time-sharing basis.

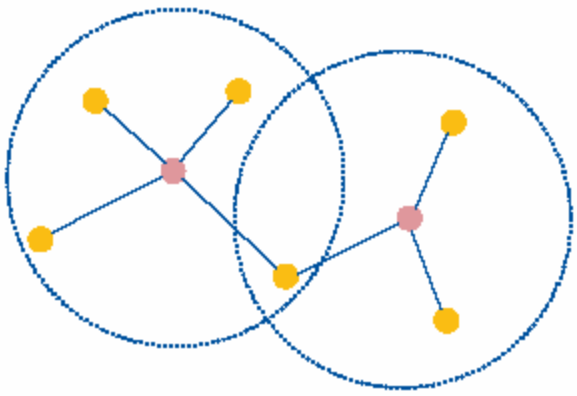

Figure 4.4 Bluetooth scatternet [16]

Bluetooth uses a procedure known as inquiry for discovering other devices; it uses paging to subsequently establish connections with them. Both inquiry and paging are asymmetric procedures. In other words, they involve the inquirer and the inquired (as well as the pager or the paged) devices to perform different actions. This implies that when two nodes set up a

connection, each needs to start from a different initial state; otherwise, they would never discover each other [16].

Bluetooth offers three different low-power modes for improving battery life:

- Hold mode is used when a device should be put to sleep for a specified length of time. The master can put all its slaves in the hold mode to suspend activity in the current piconet while it searches for new members and invites them to join.
- Sniff mode is used to put a slave in a low-duty cycle mode, whereby it wakes up periodically to communicate with the master.
- Park mode is similar to the sniff mode, but it is used to stay synchronized with the master without being an active member of the piconet. The park mode enables the master to admit more than seven slaves in its piconet.

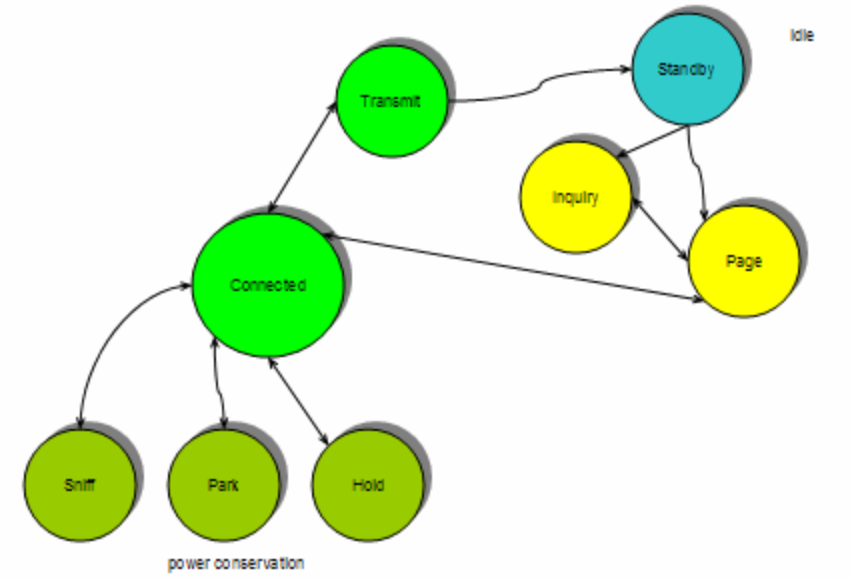

Figure 4.5 Bluetooth states.

The ultimate objective of the Bluetooth specifications is to allow multivendor applications to interoperate. Different applications may run on different devices, and each device may use a protocol stack from one vendor and a Bluetooth chip from another. Yet interoperability among applications is achieved when different implementations comply with the same core and profiles specifications [16].

4.1.2.1 Bluetooth in Palm OS

The Bluetooth APIs provide developers a way to access the Palm OS Bluetooth system and write Bluetooth-enabled applications. Bluetooth-enabled Palm OS handhelds are able to communicate with a variety of remote Bluetooth devices [15].

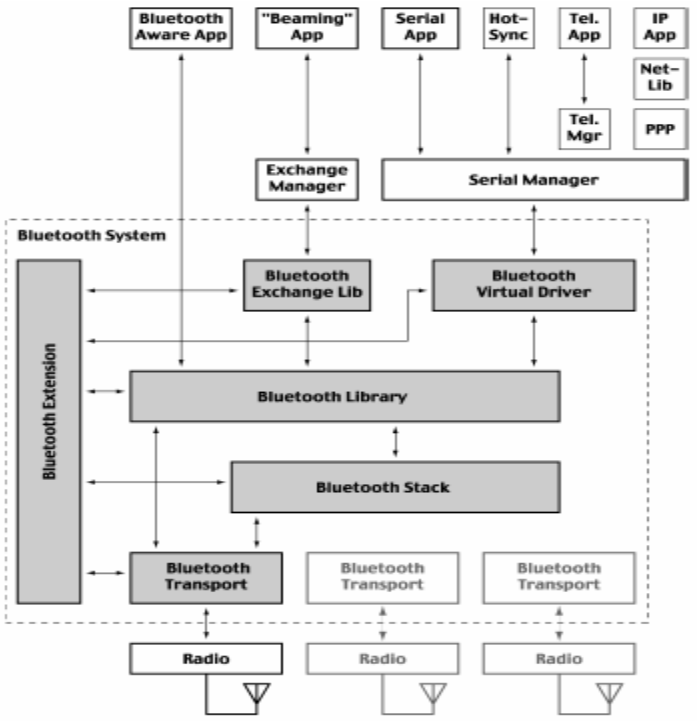

Figure 4.6 Palm OS Bluetooth architecture [9].

Functions of the Bluetooth library fall into four areas: management, sockets, security, and utility.

- The management functions deal with the radio and baseband parts of the Bluetooth specification. Developers use them to find nearby devices and establish ACL links.
- The socket functions enable communication with L2CAP, RFCOMM, and SDP.
- The security functions manage a set of trusted devices-devices that do not have to authenticate when they create a secure connection with the handheld.
- The utility functions perform useful data conversions.

Three basic management tasks common among Bluetooth applications are finding the Bluetooth devices in range, establishing ACL links, and working with piconets. However, in order for developers code to use any of the functions that do these operations, we need to create a management callback function [15].

Most management calls are asynchronous. In other words, they start an operation and return before the operation actually completes. When the operation completes, the Bluetooth Library notifies the application by way of a callback function. Such a notification is called a management event.

```
void MyManagementCallback (BtLibManagementEventType *eventP,
  UInt32 refcon) {
  switch (eventP->event) {
    case btLibManagementEventInquiryResult :
      // A device has been found. Save it in a list
      break;
    case btLibManagementEventInquiryComplete :
      // The inquiry has finished
      break;
    case btLibManagementEventInquiryCanceled :
      // The inquiry has been canceled
      break;
    case btLibManagementEventRadioState :
      // The radio state has changed
      break;
    default :
      // Unknown event
      break;
  }
```

Figure 4.7 An example of Bluetooth management callback event.

The *BtLibOpen* function is used to open the Bluetooth library. At this time, the Bluetooth library starts the radio initialization process. When initialization successfully finishes, the Bluetooth library generates a *btLibManagementEventRadioState* event with a status of *btLibErrRadioInitialized* .

There are two ways to find Bluetooth devices that are within range:

- Use the *BtLibDiscoverMultipleDevices* and *BtLibDiscoverSingleDevice* functions to find nearby devices. These functions bring up a user interface that allows the user to choose one or more devices.
- Perform a device inquiry using *BtLibStartInquiry*. This is more difficult to do than using one of the discovery functions, but provides more flexibility.

Once the application has the device address of a remote device, the application can attempt to create an ACL link to other devices using the *BtLibLinkConnect* function. This causes

the *btLibManagementEventAclConnectOutbound* event to be generated, and the status code within that event indicates whether or not the link was successfully established.

To create a piconet, the "master" calls *BtLibPiconetCreate.* Slaves can then discover the master and join the piconet, or the master can discover and connect to the slaves. The master stops advertising once the limit of seven slaves has been reached.

The Bluetooth Library uses the concept of sockets to manage communication between Bluetooth devices. A socket represents a bidirectional packet-based link to a remote device. Sockets run over ACL connections. The Bluetooth library can accommodate up to 16 simultaneous sockets.

Three types of sockets are supported by the Bluetooth Library. L2CAP and RFCOMM sockets establish data channels and send and receive arbitrary data over those channels. SDP sockets allow us to query remote devices about the services those devices provide.

To set up for inbound L2CAP connections, the application calls the following:

1. *BtLibSocketCreate*: create an L2CAP socket.
2. *BtLibSocketListen*: set up an L2CAP socket as a listener.
3. *BtLibSdpServiceRecordCreate*: allocate a memory chunk that represents an SDP service record.
4. *BtLibSdpServiceRecordSetAttributesForSocket*: initialize an SDP memory record so it can represent the newly-created L2CAP listener socket as a service.
5. *BtLibSdpServiceRecordStartAdvertising*: make an SDP memory record representing a local SDP service record visible to remote devices.

To establish an outbound L2CAP connection, first an ACL link is stabled to the remote device. Then the application calls:

1. BtLibSocketCreate: create an SDP socket.
2. BtLibSdpGetPSMByUuid: get an available L2CAP PSM using SDP.
3. BtLibSocketClose: close the SDP socket.
4. BtLibSocketCreate: create an L2CAP socket.
5. BtLibSocketConnect: create an outbound L2CAP connection.

4.1.3 Exchange Manager

The simplest form of communication for a Palm OS application is 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 [15].

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 which libraries implement each protocol and which applications receive each type of data. See Figure 4.7.

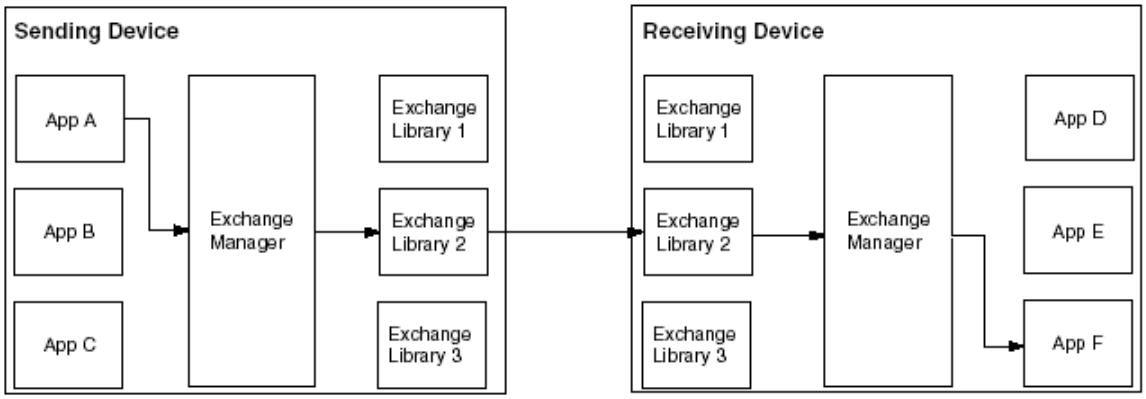

Figure 4.8 Exchange libraries [15]

Exchange Library	Minimum Palm OS Version
IR Library (IrDA)	Palm OS 3.0
Local Exchange Library	Palm OS 4.0
SMS Library (Short Messaging System)	Palm OS 4.0
Bluetooth Library[1]	Palm OS 4.0

Table 4.1 supported exchange libraries [15].

The Exchange Manager sends and receives typed data objects. A typed data object (or object) is a stream of bytes plus some information about its contents. The content information includes any of: a creator ID, a MIME data type, or a filename. The object itself can be in any format, but it's best to use a standardized data format rather than a proprietary one if you have a choice. Table 4.2 lists the standardized data formats that the built-in Palm OS applications can receive.

Application	Data Type
Address Book	vCards (vcf file extension, text/x-vCard MIME type)
Datebook	vCalendars (vcs file extension, text/x-vCalendar MIME type)
Launcher	Palm OS databases (prc, pdb, oprc, and pqa file extensions, application/x-pilot and application/vnd.palm MIME types)
Memo	Plain text (txt file extension, text/plain MIME type)
ToDo	Not explicitly registered, but receives vCalendar objects from Datebook as appropriate

Table 4.2 Built-in applications and standard data type [15].

The Exchange Manager, exchange library, and application use an exchange socket structure (ExgSocketType) to communicate with each other. This structure is passed from the application to the Exchange Manager to the exchange library and vice versa. (The use of the term "socket" in the Exchange Manager is not related to the term "socket" as used in sockets communication programming.) When an application sends data, the structure must be created and initialized with the appropriate information. When the application receives data, this structure provides information about the connection and the incoming data.

In the ExgSocketType structure the two important pieces of information are:

- The exchange library that should do the sending
- The type of data being sent

Exchange libraries are Palm OS shared libraries that act as "plugins" to the Exchange Manager. They deal with protocols and communication devices and allow Palm OS applications

to import and export data objects without regard to the transport mechanism. For example, one exchange library always available to Palm OS handhelds implements the IrDA protocol, IrOBEX. This allows applications to beam objects by way of infrared from one Palm OS handheld to another.

4.2 Implementation

We implement a framework that allows users to transparently share the record databases of common applications during spontaneous collaborative sessions. Our framework includes the following parts:

- Session Manager
- Code adapted in common Palm applications
- Policy Preferences

4.2.1 Session Manager

The first step to begin the collaboration is starting a session; the session defines which users are going to participate in the sharing process. Users need to meet some requirements in order to form part of the session: they must be in the same place where the devices are in range (no more than 10 meters of distance from each other); and they must start our application "Ad-hoc Sharing DB" (ASDB).

ASDB is an application that we developed with Palm OS APIs. It is in charge of starting the session and saves the users' policy preferences. Once the users are in a meeting, they can

start ASDB. One of the users must click the “Start Session” button (Figure 4.9), she/he is going to become the “host” of the session.

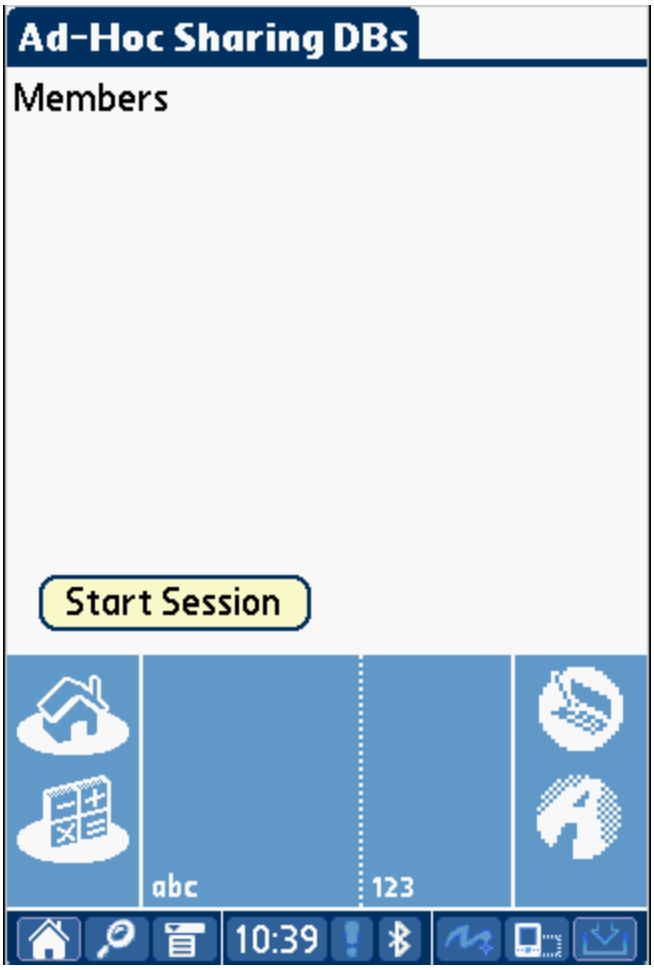

Figure 4.9 The main screen of the ASDB.

The host will look for the Bluetooth devices in range; he will select the members of the session (Figure 4.10). 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.

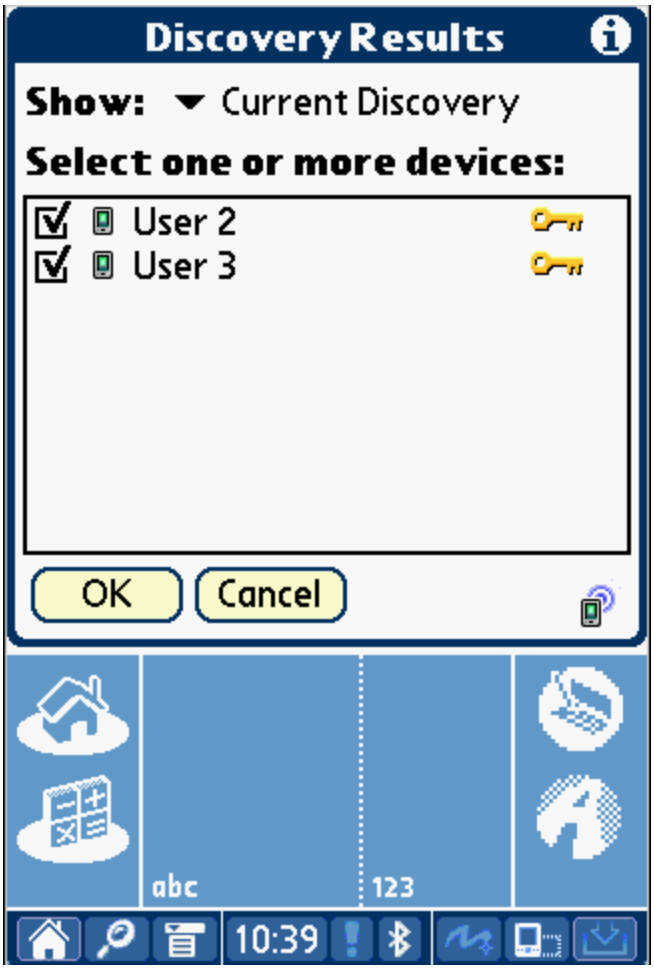

Figure 4.10 Devices in range.

The implementation of the process for creating a piconet and the process for connecting devices was programmed with the Palm Bluetooth API (see section 4.1.2.1).

The steps followed by the host device are:

- Open the Bluetooth library,
- Create a piconet,
- Discover devices in range,
- Create an ACL link to all members,
- Create a L2CAP socket to each member,
- Send all the addresses,

- Store the addresses,
- Close the socket and ACL link.

In the others' devices the processes are:

- Open the Bluetooth library,
- Wait for the host connection,
- Receive the addresses of the all members,
- Store the addresses.

Once the session is started the connection is closed in order to save the battery. After that the users can share their records using common Palm OS applications.

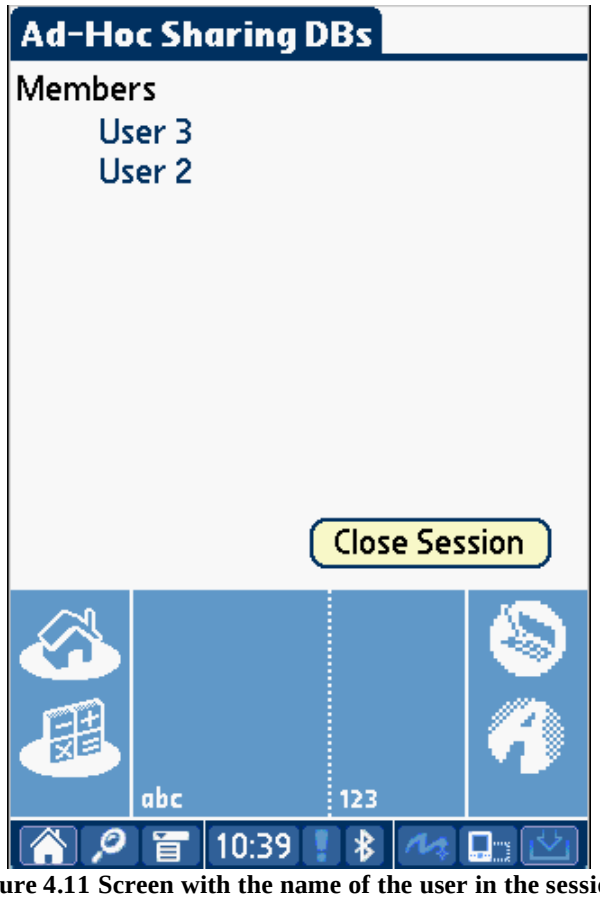

Figure 4.11 Screen with the name of the user in the session.

4.2.2 Adapting Palm OS Applications

In order to allow the collaboration by using our framework we have adapted some common Palm OS applications. The steps involved in the adaptation are:

- detect when a user inserts, modifies or deletes a record,
- notify and send the record to the ASDB application,
- send/receive the record through Bluetooth Exchange Library,
- insert, update, or delete the received record according to the user's policy preferences.

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 (Figure 4.12).

```
typedef struct {
    Char *  dbName;  //name of the data base application
    Char *  recordStr; //pointer to the beginning of the record data
    Char *  appDestName; //name of the application
    UInt16  attr; // records attribute
    UInt32  uniqueID; // unique id of the records
    UInt32  size; //size in bytes of the record
    Boolean recDeleted; // a flag if the record is deleted
    UInt16  policyAppIndex; // index of the application policies
} RecordInfo;
```

Figure 4.12 Structure to be send by the local application.

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. We are using 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 the record. Sending a launch code to another application is like calling a specific subroutine in that application.

In our framework when the adapted Palm application creates the structure with the record information, the application has to send a specific launch code to ASDB which starts the process of sending the record to the others members. The launch code is “41976”. Once the ASDB finishes the process of sending the record, the adapted Palm application takes back the control of the system.

This process is entirely transparent to the users.

```
LocalID dbID = DmFindDatabase(0,"AutoShareableDBs");
RecordInfo info;

info.dbName = dbName;
info.attr = attr;
info.uniqueID = uniqueID;
info.recDeleted = recDeleted;
info.appDestName = appDestName;
info.size = StrLen(recordStr);
info.policyAppIndex = policyAppIndex;
info.recordStr = recordStr;

if(dbID)
{
    UInt32 result = 0;
    Err err2 = SysAppLaunch(0,dbID,0,ASDBAppLaunchCmdSendData,&info,&result);
}
```

Figure 4.13 Code for sending the structure to ASDB.

When ASDB has the record information, it will send the record to the members addresses registered in the preferences. We programmed the process of sending the record using the Bluetooth Exchange Manager Library from Palm OS; we used this library to take advantage of the available technology as seen in the section 4.1.3.

In the receiver side, the ASDB will create a new structure with the same data as the received record and will add the user policy preference to it (Figure 4.14). This new structure is

passing to the corresponding application using the *SysAppLaunch* method with the launch code defined for ASDB. This process is similar to that process executed for the adapted applications when they send the record to ASDB. The launch code to call the corresponding application is "41977".

```
typedef struct Policy {
    Boolean insert; //if it is allow to insert new record
    Boolean update; //if it is allow to update the record
    Boolean deleteRec; //if it is allow to delete the record
    Boolean send; //if it is allow to send the record
} Policy;

typedef struct {
    Char * dbName; //database application name
    Char * recordStr; //pointer to the record
    UInt32  size;  //size in bytes of the record
    Char * appDestName; //name of the application
    UInt16 attr; // record's attributes
    UInt32 uniqueID; // record's unique ID
    Boolean recDeleted; //flag if the record is deleted
    Policy policy; //policies for the application
} RecordRecivedType;

typedef RecordRecivedType * RecordRecivedTypePtr;
```

Figure 4.14 Structure received by the adapted application from ASDB.

On the application side we had to modify the "pilotmain" method for catching the launch code and starting the process that stores the received record (Figure 4.15).

```
case ASDBAppLaunchCmdReciveData:
 receivedData((RecordRecivedTypePtr)cmdPBP, (launchFlags&sysAppLaunchFlagSubCall) != 0);
 break;
```

Figure 4.15 Code for processing the launch code of ASDB in the PilotMain method.

Once a record is received, the corresponding application opens its data base (if it is not open) and validates if the record already exists. The application identifies, according to the policy preferences, if the record can be inserted, updated, deleted, or has to be ignored (Figure 4.16).

```
void receivedData(RecordRecivedTypePtr recData, Boolean appIsActive){

... // code to open the database, in case it is closed.

if(!DmFindRecordByID(dbP,recData->uniqueID,&index))//update record
{
    Boolean dirty,busy,deleted;
    DmRecordInfo(dbP,index,&localAttr,NULL,NULL);
    recDirty = (localAttr&dmRecAttrDirty)>0;
    recBusy  = (localAttr&dmRecAttrBusy)>0;
    recDeleted= (localAttr&dmRecAttrDelete)>0;

    if(!recBusy)
    {
        Err error;
        ApptDBRecordFlags changedFields;

        if(recData.policy.update && !recData->recDeleted)
        {
            if(!recDeleted) //update record
                ApptChangeRecord(dbP, &index,&rec, changedFields);
            else //the record was mark for delete, record is update and unmarked for delete
            {
                DmRemoveRecord(dbP, index);
                localAttr = 0;
                ApptNewRecord(dbP,&rec,&index);
                DmSetRecordInfo(dbP,index,&localAttr,&recData->uniqueID);
            }
        } else if(recData.policy.deleteRec && !deleted && recData->recDeleted) //delete record
            DmDeleteRecord(dbP,index);
    }
}
else if(!recData->recDeleted && recData.policy.insert) //insert new record
{
    localAttr = 0;
    ApptNewRecord(dbP,&rec,&index);
    DmSetRecordInfo(dbP,index,&localAttr,&recData->uniqueID);
}

... //code to close the data base.

}
```

Figure 4.16 Code to receive and process the record.

4.2.3 Storing the Record

The adapted applications are in charge of the process of storing the record. In the adapted applications, we can define how we are going to treat the conflicts when the same record is modified by the local and remote user. The way we are handling the conflicts is by overwriting the record even if the user had changed the local record (Table 4.3).

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

Table 4.3 Rules to be applied in the received record.

Another way for handling the conflicts can be by synchronizing the changes in the records. The way in which the synchronization can be applied depends on the particular structure of the applications' records. The synchronization process can be as complex as it is in the work made in SyncML [10] and Sync [23] projects. For this project, the synchronization options are limited to the ones shown above. However future work can be extend those at a reasonable cost.

4.2.4 Policy Preferences

The policies are a set of rules defined for the user in each application. They are applied each time a record is received. The ASDB includes a policy manager that stores the rules (Figure 4.17). Users only need to define their policies one time and use them for future sessions. They also have the option to use the default policies.

There are four basic processes in which the policies are applied:

- Insert
- Update
- Delete
- Send

Each one of these processes has the following options:

- None – Ignore all the users operations.
- Host – Only accept the operations from the host (host serves as a moderator of sorts).
- All – Accept the operations from any user in the session.

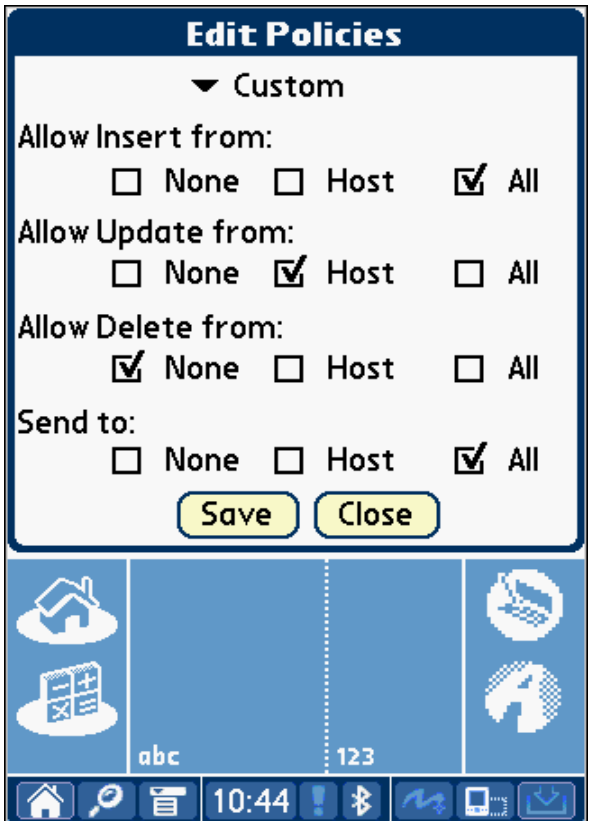

Figure 4.17 The policy editor screen.

Besides the three default policies defined in the ASDB, ASDB allows users to create their own policies in which they can set up the sharing level (Figure 4.18). Users can customize the policies for each application (Figure 4.19). These give the user the flexibility of working with different rules for each application.

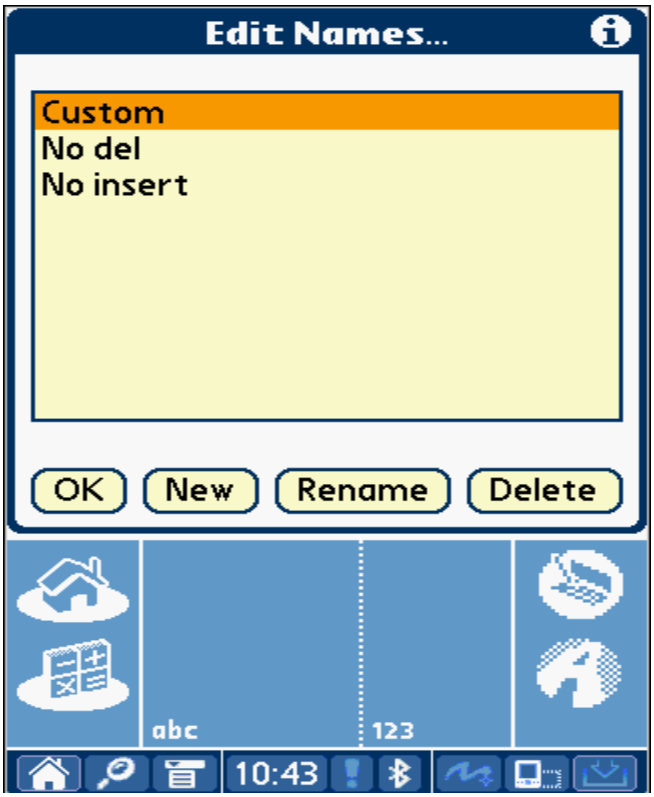

Figure 4.18 Screen for creating/renaming user's policies.

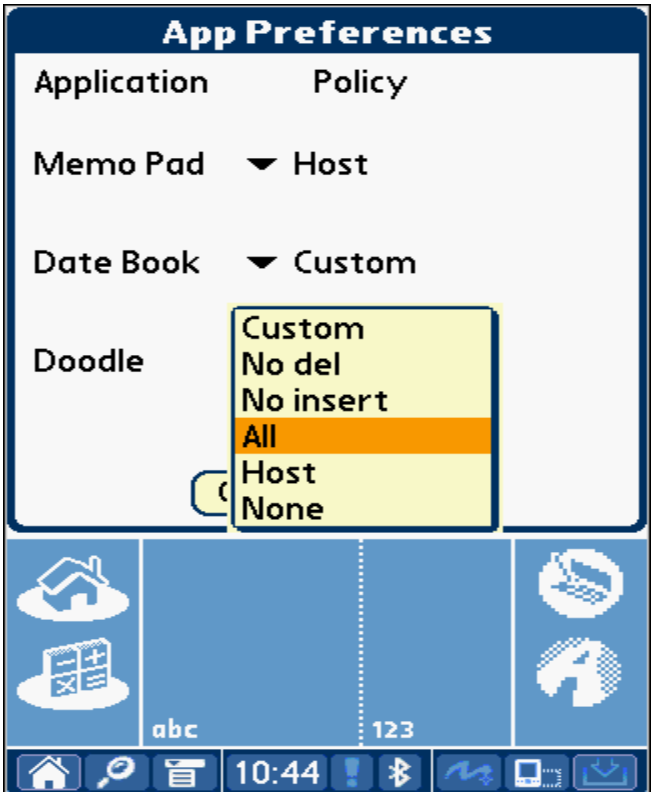

Figure 4.19 Screen for define the policy for each application.

4.2.5 Development Environment

In order to test the functionality of our framework we adapted two built-in Palm OS applications (Calendar and Memo [12, 13]) and a Sketch Notes application (Doodle [14]). With these applications we provide to users shared versions of common applications that will give them the tools to collaborate in different scenarios such as meetings.

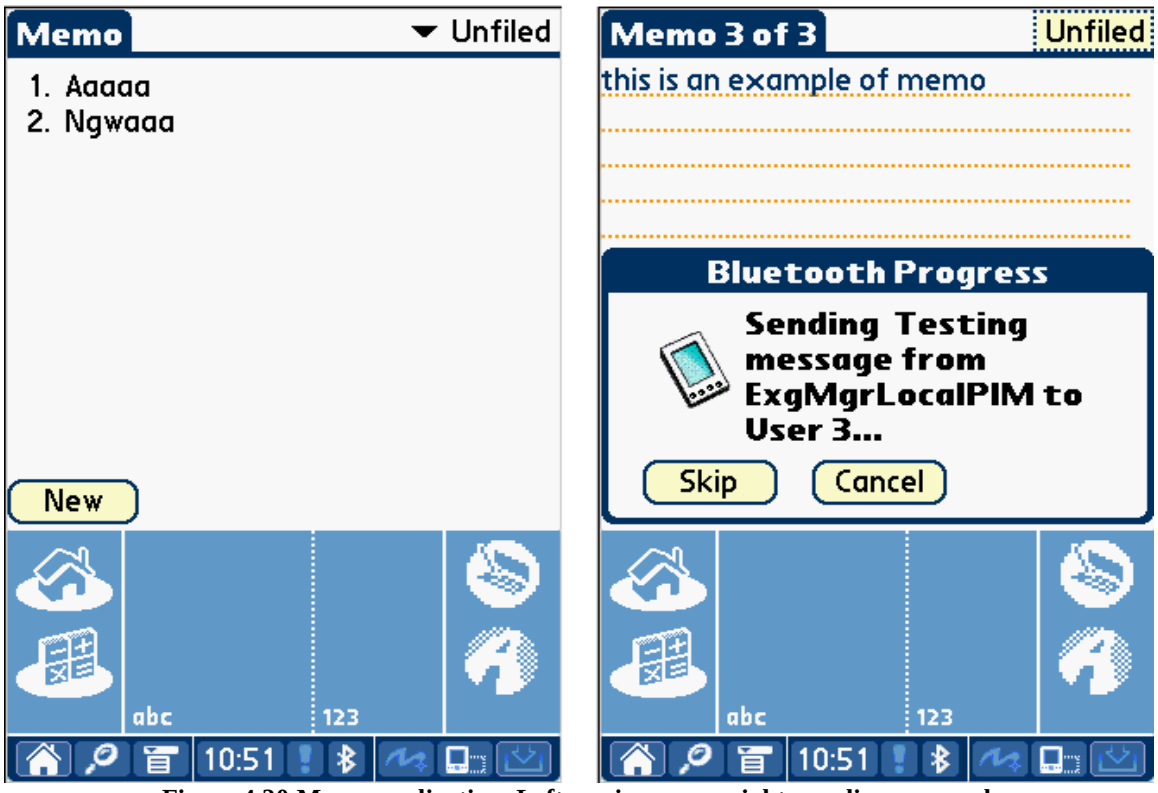

Figure 4.20 Memo application. Left: main screen, right: sending a record.

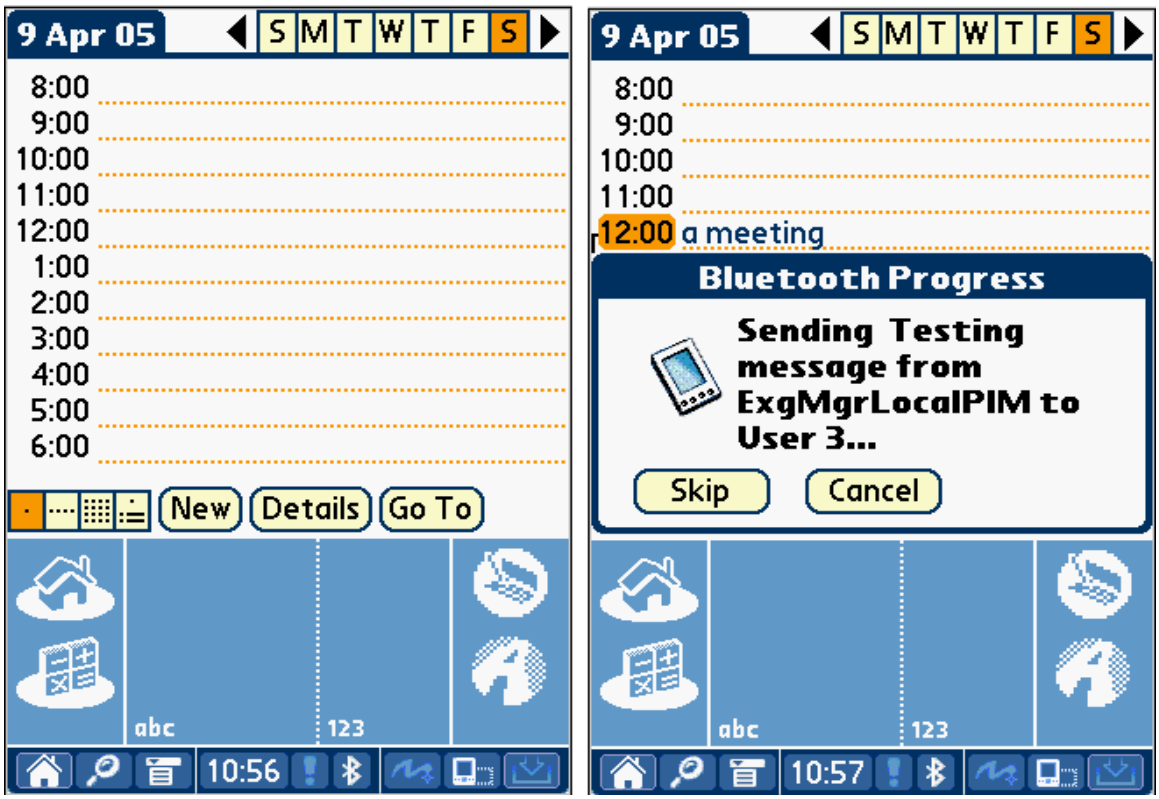

Figure 4.21 Calendar application. Left: main screen, right: sending an appointment.

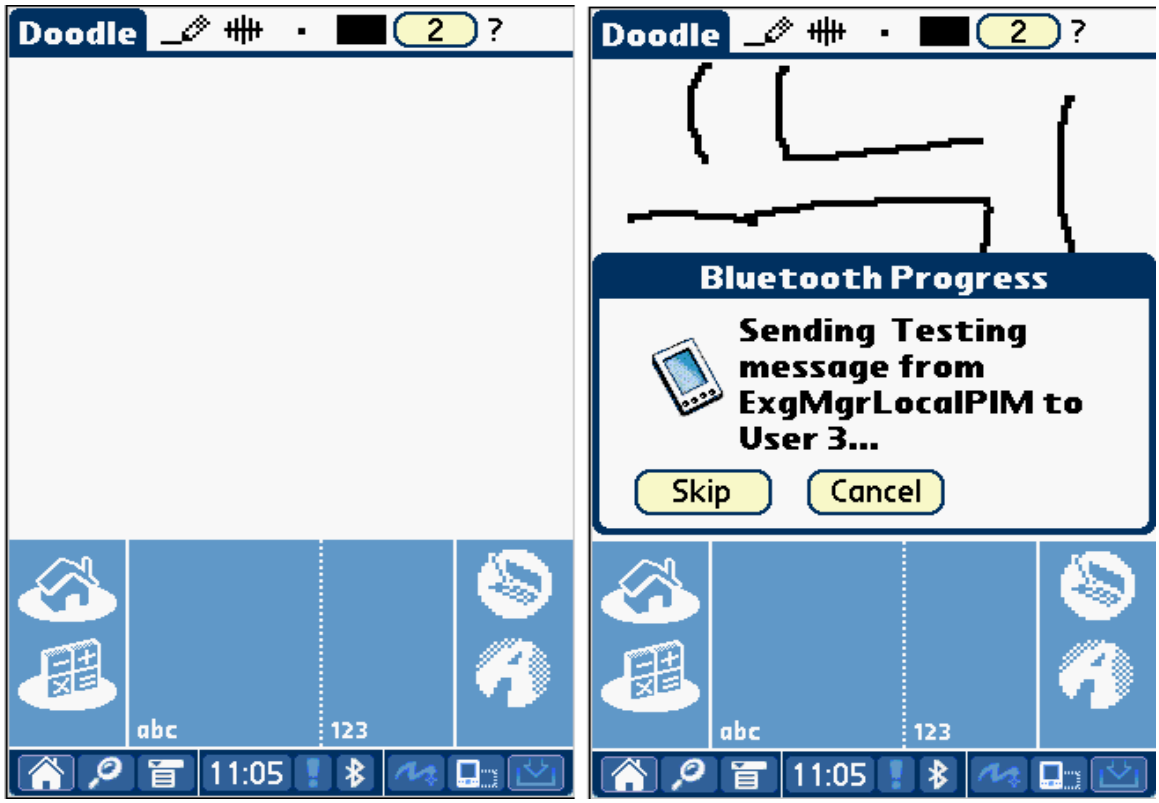

Figure 4.22 Doodle application. Left: main screen, right: sending a record.

4.2.6 Evaluation

We tested our framework in three different Palm OS handhelds, SONY Clié PEG-NZ90, Palm Tungsten T, and Palm Tungsten T3. These handhelds have different software and hardware configurations (Table 4.4).

	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

Table 4.4 Handhelds configuration.

The results that we got from our tests show that, overall, the programming effort involved is minimal relative to the size of the original application (Table 4.5, Table 4.6). Once the user starts the session and defines the policies for each application, the user interaction with the application is, essentially, the same as in the original application.

	Calendar	Memo	Doodle
# lines in the original code	28,890	6,481	2,492
# lines added	370	150	90
Percentage of adaptation	1.28%	2.31%	3.61%

Table 4.5 Development 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

Table 4.6 Complexity of code changes.

Chapter 5: Conclusion and Future Work

In this work, we presented a framework that allows Palm users to transparently share their record databases of common applications during spontaneous collaborative sessions. Users can select the members of the session without needing any pre-configuration in their devices or in a server.

With our framework, Palm developers can add the functionality of collaboration to their applications by programming a few lines of code. The integration of common Palm applications with this framework offers the users a more efficient way to share their application records without relying on a network infrastructure. This framework gives users the freedom of starting a collaborative session everywhere.

We provide users a tool to define their own policies. With the policies users determine the manner in which records are exchanged and updated. Our tool preserves the users' privacy preferences.

We tested our framework in the following Palm applications: Memo Pad, Calendar, and Doodle. We demonstrated that the adaptation of these applications was simple since the required code is minimal. The user interaction with these applications is practically the same as the original applications.

Future Work

We consider that we can extend the scope of our framework by:

- Integrating any new application to our framework without modifying the source;
- Using our framework in other devices beside Palm handhelds;
- Applying a more complex synchronization technique, when two users modify the same record.

In the actual process for integrating a new Palm application to our framework, we need to include a few lines of code in the source code of the applications. The issue with the current process is that often the source of the applications is not available. Therefore, we need an alternative that allows collaboration using those applications without the need of modifying their source code.

In this work we worked only with the Palm handhelds. However, we have realized that in the real world not all the handheld users work with Palms, sometimes they can also use Windows Mobile Edition, Linux, and even smart phones. If our framework can become platform independent, this would increase the collaboration environment. With the use of Bluetooth and synchronization protocol this limitation can be solves.

The way we are handling the conflicts, when two users update a record, is by keeping the last one modified. We consider that it would be a good option to use in our framework a synchronization technique for merging the records changes. This will give our framework more flexibility for handling the information.

These proposals will take more researching time in order to be implemented, and there is probably not an easy way to develop them. More work can be done in the area of mobile collaboration.

References

[1] Richard H. Wiggins III MD, “Personal Digital Assistants”, Journal of Digital Imaging, Vol 17, No 1 (March), 2004: pp 5-17

[2] PalmOne Historical Timeline, http://www.palmone.com/us/company/corporate/timeline.html access 2005

[3] “Personal Digital Assistants: Which One is Right for Me?” https://www.acponline.org/counseling/which_pda.htm access 2005

[4] Richard C. Davis1, James Lin1, and others. "A Framework for Sharing Handwritten Notes" UIST ’98. San Francisco, CA

[5] James A. Landay, Richard C. Davis, and others “NotePals: Sharing and Synchronizing Handwritten Notes with Multimedia Documents” University of California

[6] Jonathan Huang. “A Collaborative Property-Based Note Management System” University of California.

[7] Matthew Kam, Jingtao Wang, and others. “Livenotes: A System for Cooperative and Augmented Note-Taking in Lectures” University of California at Berkeley. ACM CHI 2005

[8] Manuel Alba, Jesús Favela. "Supporting Handheld Collaboration through COMAL" 2000 IEEE

[9] “The SyncML Initiative” http://xml.coverpages.org/syncML.html access 2005

[10] Andreas Jönsson and Lars Novak "SyncML—Getting the mobile Internet in sync"

[11] “PalmSource Palm OS Features & Benefits” http://www.palmsource.com/palmos/features.html access 2005

[12] Saul Greenberg, Michael Boyle, and others. “PDAs and Shared Public Displays: Making Personal Information Public, and Public Information Personal” University of Calgary. 1999

[13] Ivan Marsic, Allan Meng Krebs, Bogdan Dorohonceanu, and Marilyn Tremaine “Designing and Examining PC to Palm Collaboration” The State University of New Jersy. 2002 IEEE

[14] Palm OS Programmer’s Companion Volume I, Palm Source

[15] Palm OS Programmer’s Companion Volume II Communication, Palm Source

[16] Pravin Bhagwat "Bluetooth: Technology for Short-Range Wireless Apps" May-Jun 2001 IEEE

[17] Manthos Kazantzidis, Rohit Kapoor, Mario Gerla "Bluetooth – an Enabler for Personal Area Networking" University of California

[18] Palm OS Calendar source from “Palm OS 5 SDK (68K) R3”

[19] Palm OS Memo source from “Palm OS 5 SDK (68K) R3”

[20] Doodle, http://www.elf.org/doodle/index.html

[21] “Virtual File System Overview” http://publib.boulder.ibm.com/infocenter/pseries/index.jsp?topic=/com.ibm.aix.doc/aixprggd/kernextc/virtual_fsys_over.htm

[22] Andreas Jönsson and Lars Novak "SyncML—Getting the mobile Internet in sync"

[23] Jonathan P. Munson and Prasun Dewan "Sync: A Java Framework for Mobile Collaborative Applications" University of North Carolina May 1997

Vita

Gabriel Pérez was born in Villahermosa, México, in 1976. He received his Bachelor Degree in Computer Science from University of the Americas-Puebla, Mexico, in December 2000. During 2001-2003, he worked for the software industry in which he occupied different positions including manager, designer and developer.

In January 2004, he started the study in the Computer Science Graduate Program at the University of New Orleans. He completed his graduate studies in May 2005