PDAs as Remote Interactive Devices Using Native Code

Henry Berg (henry.berg@stanford.edu)

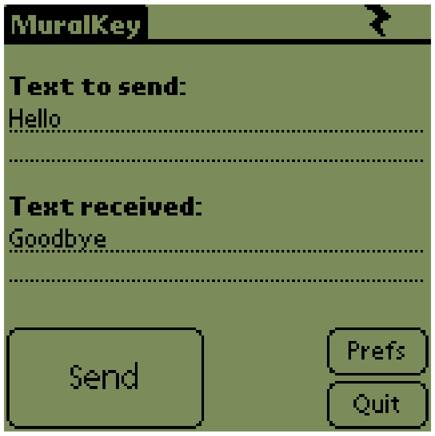
Stanford Interactive Workspaces Project

<http://graphics.stanford.edu/projects/iwork/>

Introduction

In order to study the effectiveness of PDA-native applications and interactive devices in a multi-device networked environment, we developed two applications. A remote text-entry and text display application was developed first. Once the infrastructure was in place, we went on to develop a remote control application for LCD projectors.

In order to accommodate low-latency operations, we interfaced Palm III devices to a Linux server using a direct serial connection. Palm-specific code was written in C++ using Metrowerks CodeWarrior for execution under PalmOS 3.0. For the server, standard gcc C++ was used on a PC running Red Hat Linux 5.2.



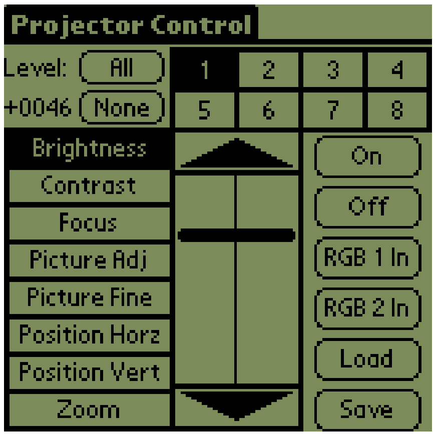


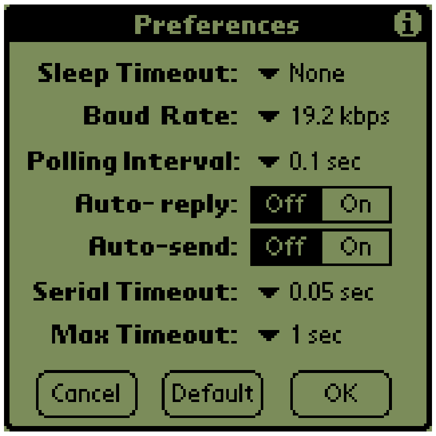
Figure 1: Main screens of the two applications.

Serial infrastructure design

The first obstacle we confronted was the lack of any existing framework for low-overhead and low-latency serial communications. Using the provided PalmOS TCP/IP stack required negotiating a PPP connection each time the Palm device was powered on or reset. We chose to develop our own serial protocol on top of an implementation of RPC Record Marking Standard (RFC-1831), a simple serial packet protocol.

C++ libraries were written to support sending and receiving packets on both the Palm device and Linux server. The Palm device library was written to power-down the serial port after each packet, allowing use after negligible delays regardless of reset status. For extreme low-latency applications, the port could optionally remain powered-up.

PalmOS 3.0 is, from the developer's point of view, a single-threaded event-based operating system. Palm applications operate by properly responding to existing events and generating new events. To allow applications to receive data without explicitly polling the serial port, an event-based serial library was constructed. This allowed the application to respond differently to serial communications in different parts of the application, without duplicating all of the serial code. Applications using this library handled all of their serial communication via a customized series of events. Polling the port for waiting data was handled behind the scenes by the serial event library. When serial data was waiting for receipt, an event was generated. Applications wishing to respond to unsolicited serial transmissions needed only to respond to this event and collect the data.



Sleep Timeout:	None
Baud Rate:	19.2 kbps
Polling Interval:	0.1 sec
Auto-reply:	[✓ Off] On
Auto-send:	[✓ Off] On
Serial Timeout:	0.05 sec
Max Timeout:	1 sec

Figure 2: Serial and event preferences from the text application.

One benefit of our standardized packet protocol was that the application need know nothing about what was on the other end of the serial connection. With the text application, using a null modem cable to connect two Palm docking cables allowed two Palm devices to send data back and forth.

Several problems were also encountered, some unresolved. Limitations of the Palm device and PalmOS 3.0 limited communications to 19,200 baud. RS-232 connections are totally intolerant of mismatched communications parameters, so some sort of standard fallback set might be added to allow connection negotiations. One of our Palm devices had a faulty UART chip (since repaired), resulting in dropped data. This resulted in loss of synchronization, which our current library has no way of handling, other than re-sending the entire packet.

Customized interactions with native code

Once communications support was established, we developed the Projector Control application to experiment with different ways of interacting with native applications. While the text application was written using only widgets built into PalmOS, the central widget in the projector application was a flexible slider.

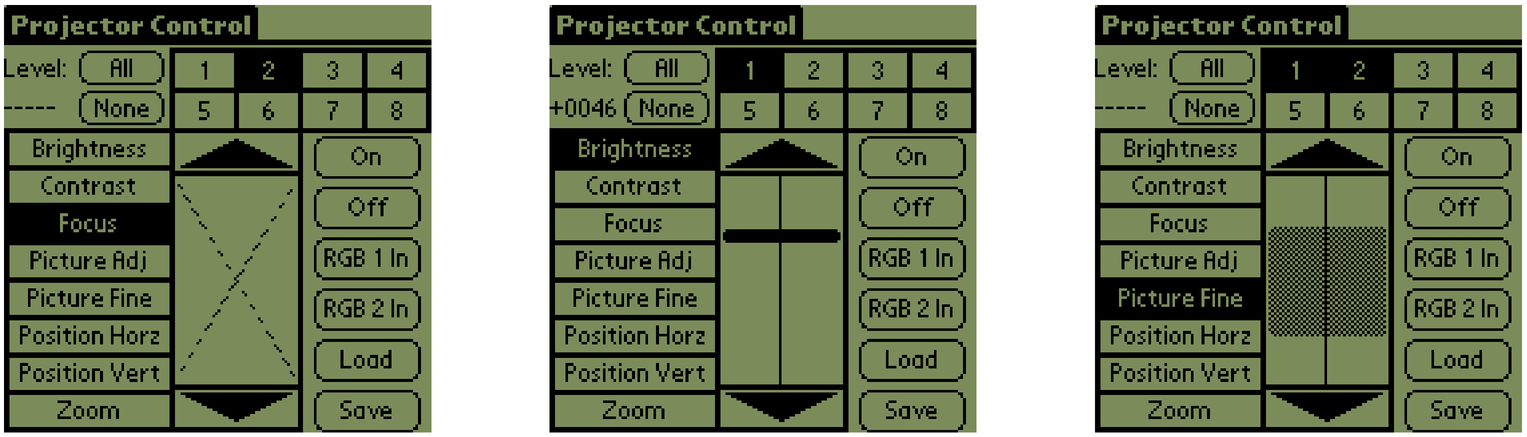


Figure 3: Various slider configurations (see text).

Three screen shots from the projector application are shown in figure 3. A projector attribute was selected from the left side of the screen along with one or more projectors from the top. The arrow buttons and slider control are then used to adjust that attribute for the projector(s).

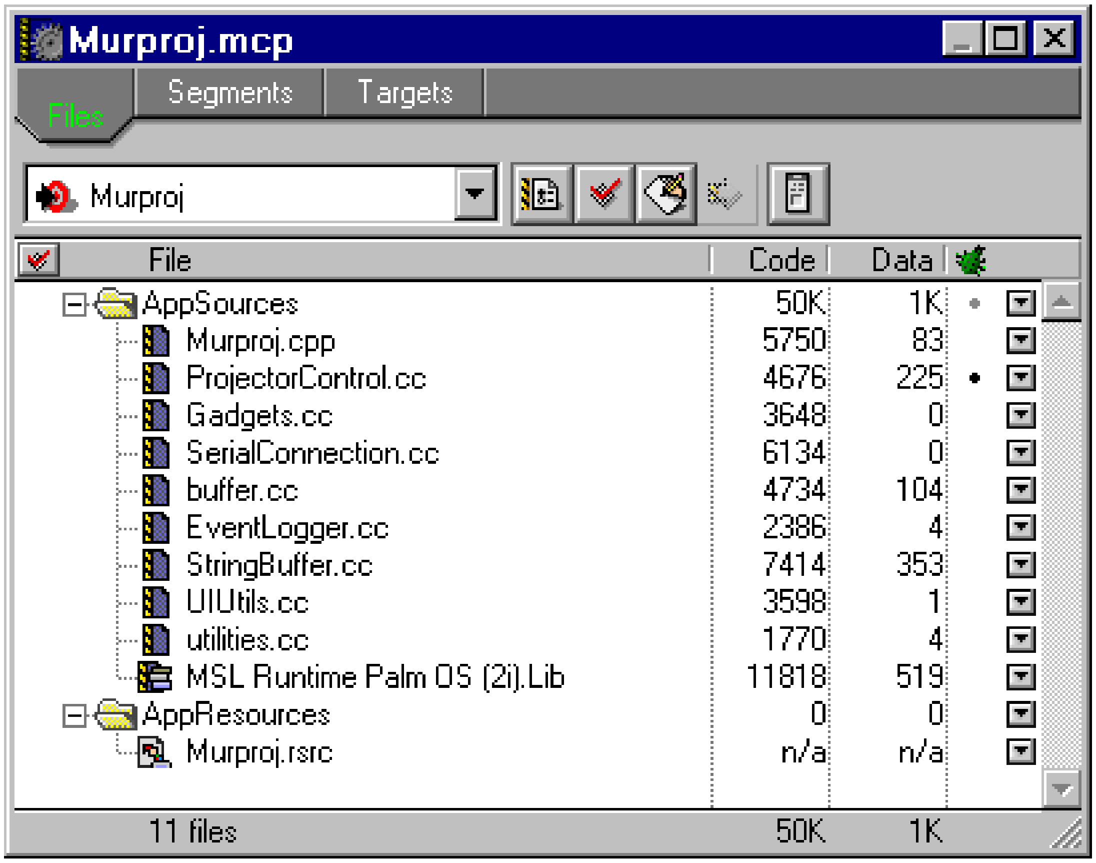
In some cases, due to the design of the projector, only arrow button adjustment was allowed. In these cases, the slider X'd itself out, and only the arrow buttons could be used. When a value was known, a single slider bar was displayed, with the level that represented in text at the upper-left hand corner of the screen. This value was updated as the slider was dragged up and down by the user.

When the user had selected an attribute and more than one projector, the range, if any, in the selected attribute was displayed as a gray box. Touching anywhere in the slider range would set all selected projectors to the new value.

By developing our own slider widget, we had detailed control over its interactive parameters. This was a mixed blessing, as we were forced to define such things as the exact pixel range for slider handle stickiness, how far the pen could move before releasing the slider, how to interpolate between attribute and pixel ranges and the exact patterns used for corners.

Conclusions

There are many types of interaction support that require low-level control over both the user interface and communication method used. We choose the projector control application as one where a traditional network connection and standardized Palm widgets were simply not up to the task.



File	Code	Data
AppSources	50K	1K
Murproj.cpp	5750	83
ProjectorControl.cc	4676	225
Gadgets.cc	3648	0
SerialConnection.cc	6134	0
buffer.cc	4734	104
EventLogger.cc	2386	4
StringBuffer.cc	7414	353
UIUtils.cc	3598	1
utilities.cc	1770	4
MSL Runtime Palm OS (2i).Lib	11818	519
AppResources	0	0
Murproj.rsrc	n/a	n/a
11 files	50K	1K

Figure 4: The CodeWarrior project window for Projector Control.

However, the complexity involved with this kind of detailed control is somewhat daunting. Figure 4 shows the CodeWarrior window from the projector application. Even with as much complexity as possible abstracted into C++ library files, PalmOS remains a programmer-hostile environment. We are currently working on ways to allow a blend of low and high-level solutions to make it easier to develop PDA-based low-latency interactive applications.