MARS: Portable Support for Community-Based Research Using Cellular Phones

Maria Kazandjieva
Stanford University
mariakaz@stanford.edu

Margaret Martonosi
Princeton University
mrm@princeton.edu

ABSTRACT

With over 2 billion cellular telephones in daily use around the world, cell phones in just over 30 years have moved from technological oddities to a major force in the computing world. Because of their ubiquity, cellphones represent a natural platform for enabling community-based participatory research. Unfortunately, although they outnumber desktop platforms by at least a factor of two [3, 2], their software model is considerably more proprietary, including operating systems and non-standardized APIs. The closed nature of cell phone programming environments constrains their use as computing platforms, and inhibits the development of efficient, interoperable, collaborative or third-party software.

In this work, we propose the MARS framework to address these issues of efficient and portable software interoperability. MARS allows a portable upper layer of software (written in Java, and portable to any device running Java CLDC), to communicate with local or remote devices in a general but efficient manner. Generality is achieved by implementing sockets, so that device-independent Java code can communicate with device-specific stubs in order to access local device-dependent services like cameras. Efficiency is achieved by implementing "short-cuts" that turn socket accesses into local file accesses. Through this method, MARS enables portable code to access local services (e.g. battery levels or camera commands) in a general manner, while also improving communication latencies by a factor of at least 25X for a 100kB chunk of data. Overall, we see this work as an important step towards providing general layers for portable cell phone programming.

1. INTRODUCTION

In recent years cell phones have become a powerful computing platform used by millions of people. The cell phone market has particularly surged in developing countries in Asia and Africa. In India, for example, we see a growth on the order of 6 million new cell phone subscribers per month [8]. The popularity of mobile handsets in such regions is partially due to their affordability (compared to that of traditional desktops) and due to their ease of use in areas without a traditional wired infrastructure.

Widespread mobile infrastructure is a fundamental enabler of community-based participatory research (CBPR). CBPR can be used to involve regular citizens with research projects and collaborative learning in an active and empowering way. For example, CBPR has been encouraged in health-related research [1] by giving community members the chance to share observations on health and disease issues with the primary investigators on a research project. Our work improves on the software infrastructure that makes this sort of wide spread participation possible.

CBPR involves effort on both sides—researchers and community members—and as such requires adequate means of communication between them. While cellular phones are an obvious choice for this, they are not currently sufficient to use for computation and data sharing in addition to voice communication. First, software deployed for CBPR must run on a variety of platforms. We would like to maximize code portability between platforms in order to reduce the development overhead. On the community side of a project, software for CBPR should give users the option to opt out at any moment if it is inconvenient, if batteries are low, etc. CBPR must also work with extreme heterogeneity, since it is impossible to expect all participants sharing health or crop information, for example, to have the same phone. Leaving considerable choice in the participants' hands makes them more independent and provides a greater potential for empowerment [5].

A fundamental tension exists between the exciting fact that cell phones are an incredibly widespread programmable computing platform today, and the unfortunate fact that cell phone programming is constrained by proprietary, non-portable software and difficult, often-nonfunctional APIs. How, then, can a software developer interested in CBPR create a general program that runs on a myriad of heterogeneous devices, that allows users to opt in or out, and that despite its generality is also capable of accessing device-specific features such as cameras and battery gauges? In most cell phones today, such devices are only accessible through APIs that call into proprietary operating systems such as Symbian [7] or Windows Mobile [9]. Furthermore, these APIs are often only available through particular programming languages like Symbian C++ (for Symbian) or C/C# (for Windows Mobile). This paper describes MARS, an interface layer for improving the portability and efficiency of cellular telephone software. MARS lets the programmer split

SenSys'07, November 6–9, 2007, Sydney, Australia.

code into a system-specific and a system-independent layer. Therefore, a system for gathering health vaccination data could run on a number of different cell phones, reusing the system-independent code, while also taking into account the device-specific aspects of each handset.

By writing code using MARS, CBPR can ensure that every person can make an individual choice about their participation. Furthermore, the upper layer of MARS code will allow a common interface to software regardless of handset. This is especially important in areas of low literacy, where users might have difficulties adjusting to new GUIs. Providing a unified look and feel to an application and leaving the choice of hardware to the participant increases the likelihood of CBPR.

Of course, we have already run into the problem of code interoperability in the desktop computing world. Java for example, provides programmers with the Java Native Interface (JNI) which allows Java programs to call system-specific methods written in C. JNI however has not been implemented for cell phones. In addition, there are a number of interprocess communication methods that ensure data exchange between heterogeneous applications on desktops. These include shared memory, message passing, and network sockets. While sockets provide an easy way for programs written in different languages to exchange data, we show in later sections that straightforward, naive implementations are unacceptably slow.

This paper proposes a solution to the aforementioned problems on cellular phones. We have implemented MARS (Mobile Application Replacement Sockets) to provide the necessary interprocess communications between generic portable upper layer software and the proprietary low level device interfaces. In the general case, MARS implements communications via sockets. In local connections, however, these sockets are transparently translated into file-based communications to make them efficient. Thus this work accomplishes the following:

- We design an efficient layered programming interface for balancing the needs for both portability and device-specificity in mobile phones.
- To make our interface practical and efficient, we have built the MARS sockets communications system with transparent optimizations for local connections. MARS sockets can be used instead of or in conjunction with traditional sockets for maximum flexibility
- Our prototype implementation demonstrates the viability of our approach. It runs on a Cingular 8525 hand set and is written in Java and C# for Windows Mobile platforms. Our sockets optimization offers speedups of 25X over standard sockets for 100kB data transfers on this platform.

The rest of this paper is organized as follows. Section 2 provides a system overview and experiential results. Section 3 includes a detailed case study to illustrate the need for MARS. Finally, Section 4 concludes our work.

2. SYSTEM OVERVIEW AND RESULTS

In this section we provide an overview of the MARS communication infrastructure.

As shown in Figure 1 MARS sockets use the underlying file system to transfer information between two processes. Unlike standard sockets that transfer data through the network interface, MARS uses files. In order to keep interfaces simple, we only make minor modifications to the already existing Socket/ServerSocket objects in Java and C#. We create a layer of abstraction such that the programmer can write almost identical code regardless of whether she wants to use remote sockets or MARS sockets for local communication. This allows the programmer to take full advantage of regular sockets through MARS. Figure 2 shows how opening a remote connection differs programmatically from communication on the local node. Depending on how the hostname and port number are specified when a new MarsSocket object is declared, the underlying system can deduce whether standard sockets or files are to be used. This undercover detection allows the software programmer to write identical code for local and remote connections after the socket has been established.

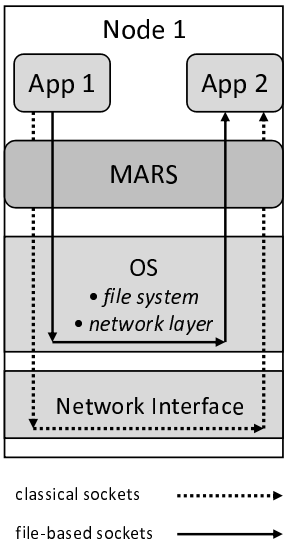

Figure 1: MARS provides a shortcut for local communications by using the file-system instead of the physical network interface.

There are a number of steps a program must follow when using MARS local sockets. Below we give a brief description of the processes that take place.

Application Registration For applications to communicate with each other, a central facility keeps track of application names and IDs. Therefore, when a cell phone is booted up, all applications using the MARS infrastructure register. This registration requires writing an application name to the *regfile* and receiving a unique ID. Since the *regfile* is a global structure we provide locking and file-sharing mechanisms to ensure there is no concurrent file access.

Peer Application Lookup Once a program is ready to establish a local connection, it performs a peer ID lookup in the registration file by providing a process name. This is an additional step that is not present when standard sockets, and thus known port numbers, are used. This difference is shown in Figure 2. Once the peer ID has been discovered, however, the process continues identically for both local and remote connections.

Initiating Connection The *commfile* it keeps track of applications requesting to open local connections. Each request is represented by a "hello" message including the source

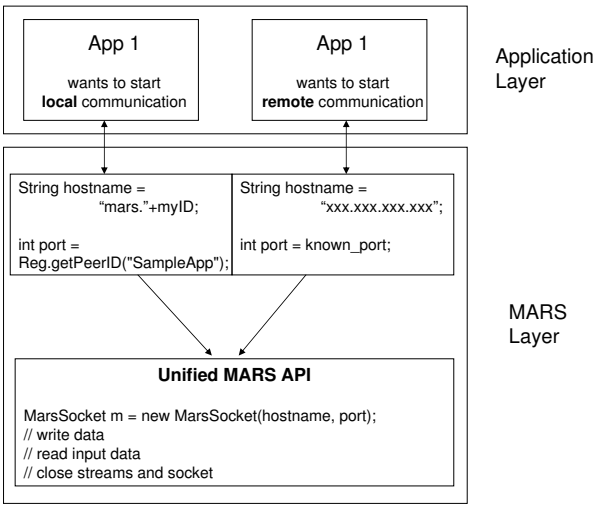

Figure 2: MARS can be used for local as well as for remote communications. Here we show what code differences a programmer will experience depending on the context in which MARS is used.

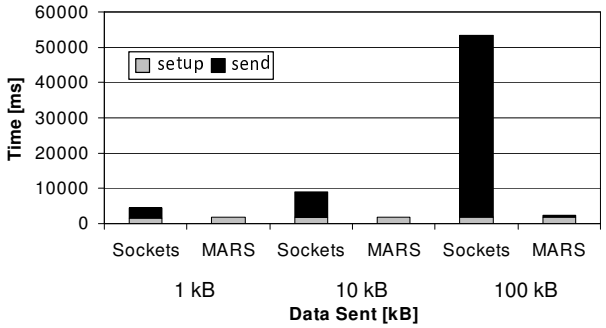

Figure 3: A comparison of MARS and regular sockets sending varying amounts of data.

and destination application IDs. A client program writes a communication request, while a server periodically checks the file for outstanding messages. These two actions are equivalent to the remote socket *listen()* and *connect()* methods for in Java.

Communication Period Once an application discovers a "hello" message, the two peers proceed to exchange data. The exchange of data does not require the programmer to know whether communication is local or remote. All that is necessary is the declaration of the MarsOutputStream and MarsInputStream objects, used for sending and receiving data respectively. Under the cover, MARS determines if data is to be sent using a network connection or simply written to a file. MARS has a file naming convention that allows applications to easily detect the read and write files without any effort on the programmer's side.

Every application-to-application local communication results in the creation of two files. Each one is a data input stream for one of the peers and a output stream for the other peer. Therefore, if an application want to communicate to two peers, it will create two different output files, and will read from two different input files. Again, this is analogous to using different port numbers for different communication pairs. Allowing only one application to write to a file in a given communication round eliminates many of the contention and locking issues we would see if two applications were sharing one file.

As in the case of network communications, we assume that the applications that are exchanging data have previously agreed on a protocol. Therefore, MARS allows the program to block waiting for input. At the end of a communications period, each application is expected to send a *"goodbye"* message, indicating that communication files are ready to be deleted. Removing files at the end of a successful connection ensures that the required storage space is freed up.

Results We developed MARS for use with Java and C# programs and implement and deploy it on a Cingular 8525 smart phone running Windows Mobile 5.0. The phone has a 400MHz processor, 128Mb of ROM and 64 Mb of RAM. We test the efficiency of the MARS interface by timing transmissions for different amounts of data. A client initiates a connection, sends the corresponding number of bytes and waits to receive an "OK" acknowledgement from the server. Figure 3 shows our results. The setup time includes time needed to create client and server sockets, obtain application IDs, open any necessary files, complete a TCP/IP handshake, and so on. The transmission time, on the other hand, is the time for sending all the bytes over and receiving the acknowledgement. Both sockets and MARS have a steady setup overhead regardless of the data size - between 1700 and 1800 ms. The communication speed, however, differs dramatically between the two methods. Sending as little as 1kB of data benefits from a 2X communication latency improvement when MARS is used and the savings go up to 25X or more for sending 100kB. Socket communication via the loopback interface using TCP/IP involves the operating system as well as the underlying physical network interface, thus affecting negatively the speed of communication. On the other hand, cell phones use flash-based storage systems, therefore making writing and reading data fast. In addition, MARS operates at the OS level resulting in addition speedups. Overall, the time savings provided by MARS increase without a bound after the one-time setup period is completed.

3. CASE STUDY: RESOURCE-AWARE CAMERA USAGE

Previous work has looked at economic models for resource sharing in wireless networks [4]. For example, in a network of mobile phones, one handset might request to use another one's camera. Taking advantage of this idea and existing wireless infrastructure, farmers in rural areas could request photos of crops from each other. This would enable them to have up-to-date information from various locations where good are grown. We would like provide every cell phone with a program that determines if there is sufficient energy to take a photo, and if so, take a snapshot and send it to another phone. We could simply write one large program responsible for monitoring system status, accessing camera to take pictures, as well as communicating with other nodes. This is feasible if we assume that all participating phones are running the same operating system. However, in the more likely

Windows Mobile .NET C#

```
\\ obtaining battery information
BatteryLevel batteryLevel =
SystemState.PowerBatteryStrength;

BatteryState batteryState =
SystemState.PowerBatteryState;

\\ capturing an image
CameraCaptureDialog ccd = new
CameraCaptureDialog();
ccd.Mode = CameraCaptureMode.Still;
ccd.Resolution = new Size(160, 120);
ccd.StillQuality =
CameraCaptureStillQuality.High;
ccd.ShowDialog();
Show();
fileName = ccd.FileName;
photo.Image = new Bitmap(fileName);
```

Symbian C++

```
\\ obtaining battery information

iBatteryStrengthIndex =
iBatteryInfoV1.iChargeLevel

iBatteryInfoV1.iChargeLevel;
iState = EStateChargerInfo;

\\ capturing an image

iState = ESnappingPicture;
iCamera->StopViewFinder();
iCamera->CaptureImage();
```

lower layer, system-specific code

Figure 4: Since cell phones run on proprietary operating systems and access system-level information through proprietary languages, one needs to rewrite system-specific code for various platforms.

```
1: int peerID = CommunicationReg.getPeerID("ProccessName");
2: MarsSocket local_m = new MarsSocket("mars." + myID, peerID);
3: MarsInputStream local_in = local_m.getInputStream();
4: battery = local_in.read();
5: image_filename = local_in.read();

6: MarsSocket remote_m = new MarsSocket("168.126.8.11", 1337);
7: MarsOutputStream remote_out = remote_m.getOutputStream();
8: remote_out.writeBytes(new ImageObject(image_filename).getBytes());
```

MARS local and remote communications

Figure 5: The MARS APIs provide a unified way to establish both local and remote connections. Where possible local connections are transparently converted to file accesses, thus decreasing communication latencies.

case, users will own cell phones with the Windows Mobile OS, Symbian OS, Linux, and even Openmoko [6]. Thus we would like to reuse code as much as possible, while still allowing system-specific operations. Therefore a MARS-based approach is ideal; an upper level code will be used for communication between devices and system-independent tasks, while a lower level will be used to access the system-specific information (camera availability and battery level) and devices such as the camera itself.

We implement this example on the Cingular 8525 hand set and use our MARS prototype to enable fast communications. Figure 4 shows code snippets for obtaining battery information and taking a picture on Windows Mobile OS and Symbian OS. This is the lower level code which is system-specific and needs to be rewritten for different operating systems. Then in Figure 5 we present MARS code that is partially used for local communications on both the upper and lower level of software, as well as for remote communication on the upper, Java level. In Figure 5, lines 1 through 5 show how an application obtains a peer ID, creates a new local socket (transparently translated to file access), and obtains data from its peer local application. Lines 6 through 8 show how we create a remote socket and send a photo, byte by byte.

Overall, we have achieved two main goals:

- code has been successfully divided into an upper, system-independent layer and a lower, system-specific layer allowing code reusability and system interoperability
- the APIs for local and remote communications have been unified by using MARS sockets, while transparently converting local connections to file accesses that improve communication latencies.

In this example, the data sent using local MARS communications is about 100 bytes (battery and camera information). In Figure 6 we see how the speed of communication compares depending on whether MARS or regular sockets are used. As expected, MARS performs better, showing a 2X improvement over standard sockets via localhost. The setup time for both types of connections is about the same, therefore, it is the usage of files that causes the speedup in communication latencies. An application that monitors

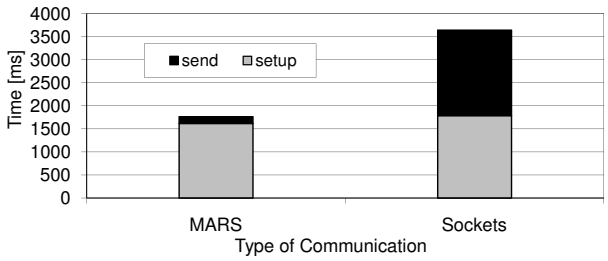

Figure 6: For the system monitoring example, MARS communication latencies are roughly one half that of the sockets implementation.

system status in a similar way but on regular basis will benefit from MARS even more because once setup cost is paid, regular fast data transmissions will keep bringing additional time savings.

4. CONCLUSION

This paper explores the design and implementation issues of MARS, an interprocess communication method aimed at cellular phones. MARS allows us to connect processes written in different languages and thus provides the ability to separate code into system-specific and system-independent counterparts. The separation into lower and upper level code promotes reusability of the upper layer on various cell phone and desktop operating systems. Our experiments show that for a 100kB chunk of data sent from a C# to a Java application, communication times are 25 times faster using MARS instead of regular network sockets. The MARS APIs make the transition from sockets easy by unifying method calls for remote and local connections. We believe that a fast and efficient method for local communications will allow re-usability of code and foster the creation of new application to be deployed on variety of cellular phone.

5. REFERENCES

[1] Community Based Participatory Research Conference Summary, 2001. http://www.ahrq.gov/about/cpcr/cbpr/.

[2] CIA world factbook. https://www.cia.gov/library/publications/the-world-factbook/, 2007.

[3] iMedia Connection. Cell Phone Usage Continues to Rise, Apr. 2002. http://www.imediaconnection.com/news/581.asp.

[4] Kazandjieva, M., and Martonosi, M. Lightweight Economic Models for Resource Sharing in Wireless Networks, Mar. 2007. Poster. In Proc. ACM SIGCSE.

[5] Learner Voice, 2007. http://futurelab.co.uk.

[6] OpenMoko. http://www.openmoko.org/, 2007.

[7] Symbian Developers Network. http://developer.symbian.com, 2007.

[8] Timmons, H. For the rural poor, cellphones come calling, 2007. http://www.iht.com/articles/2007/05/06/business/wireless07.1-44394.php.

[9] Windows Mobile Developer Center. http://msdn2.microsoft.com/en-us/windowsmobile, 2007.