Smart Phone: An Embedded System for Universal Interactions

Liviu Iftode, Cristian Borcea, Nishkam Ravi, Porlin Kang, and Peng Zhou
Department of Computer Science
Rutgers University, Piscataway, NJ 08854-8019
{iftode, borcea, nravi, kangp, pzhou}@cs.rutgers.edu

Abstract

In this paper, we present a system architecture that allows users to interact with embedded systems located within their proximity using Smart Phones. We have identified four models of interaction between a Smart Phone and the surrounding environment: universal remote control, dual connectivity, gateway connectivity, and peer-to-peer. Although each of these models has different characteristics, our goal is to design and implement an architecture that provides a unique framework for all of the models. Central to our architecture are the hybrid communication capabilities incorporated in the Smart Phones. These phones have the unique feature of incorporating short-range wireless connectivity (e.g., Bluetooth) and Internet connectivity (e.g., GPRS) in the same personal mobile device.

1 Introduction

Recent advances in technology make it feasible to incorporate significant processing power in almost every device that we encounter in our daily life. These embedded systems are heterogeneous, distributed everywhere in the surrounding environment, and capable of communication through wired or wireless interfaces. Ultimately, the large scale deployment of such systems will lead to a computerized physical world with which we can potentially interact faster and in a simpler fashion.

People, however, are not yet taking advantage of this ubiquitous computing world. Despite all the computing power laying around, most of our daily interactions with the surrounding environment are still primitive and far from the ubiquitous computing vision [25]. Our pockets and bags are still jammed with a bunch of keys for the doors we have to open/close daily (they did not change much since the Middle Ages), the car key or remote, access cards, credit cards, and money to pay for goods. Any of these forgotten at home can turn the day into a nightmare. If we travel, we also need maps and travel guides, coins to pay the parking in the city, and tickets to take the train or subway. In addition, we are always carrying our mobile phone, which for some mysterious reason is the least likely to be left at home. When we finally arrive home or at the hotel, we are “greeted” by several remote controls eager to test our intelligence. All these items are absolutely necessary for us to properly interact with our environment. The problem is that they are too many of them, they are sometimes heavy, and we will likely accumulate more and more of them as our life goes on, requiring much larger pockets.

For a number of years, visionary papers have presented a picturesque world in which all the annoying effects of modern society will go away [25, 22]. Pervasive/ubiquitous computing offers a great model for the future of the world, but very few of the applications that were anticipated have made their way into the mainstream of our daily life. In our opinion, this is due to two factors. The first is a lack of synchronization between the proposed theoretical solutions and the availability
or maturity of various supporting technologies. The second is a mismatch between the inherent complexity of the IT solutions and the simplicity expected by the end-user from these technologies.

For the problem mentioned above, the community does not lack innovative solutions to address some aspects of it (e.g., wireless microservers [19], electronic payment methods [14, 6], or digital door keys [17]). What is missing is a simple, universal solution, which end-users are likely to accept easily. Ideally, we would like to have a single device that acts as both personal server [24] and personal assistant for remote interaction with embedded systems located in proximity of the user. This device should be programmable and support dynamic software extensions for interaction with newly encountered embedded systems (i.e., dynamically loading new interfaces). To simplify its acceptance by society, it should be a device that is already carried by people wherever they go.

We believe that Smart Phones are the devices that have the greatest chance of successfully becoming universal remote controls for people to interact with various devices from their surrounding environment; they will also replace all the different items we currently carry in our pockets. Smart Phone is an emerging mobile phone technology that supports Java program execution and provides both short-range wireless connectivity (Bluetooth) and cellular network connectivity through which the Internet can be accessed. Far from being a trivial addition, the embedded dual connectivity, along with a reasonable computing power, makes the Smart Phone the long awaited universal personal assistant that can make our daily life much simpler. Examples of Smart Phones are the Sony Ericsson P800/P900 [7] and Motorola A760 [11].

The idea of using a single device for everything is not new. There have been several pioneering projects both in academia and in industry, suggesting the use of a PDA or of a mobile phone as a personal or micro server [19, 24, 17]. Solutions to a particular problem from the list presented above have also been proposed. To the best of our knowledge, however, none of the previous work has approached this problem in a systematic way, proposing a unified architecture for all these applications, and taking the risk to implement it on real devices. This delay is due in part to the fact that Smart Phone technology is very new (Ericsson P800 was introduced in March 2003, while Motorola A760 in August 2003).

In this paper, we present a system architecture that allows a user to interact with embedded systems located within her proximity using a Smart Phone. We have identified four models of interaction between a Smart Phone and the surrounding environment: universal remote control, dual connectivity, gateway connectivity, and peer-to-peer. Although each of these models has different characteristics, our goal is to design and implement an architecture that provides a unique framework for all the models. Central to our architecture are the hybrid communication capabilities incorporated in the Smart Phones. These phones have the unique feature of incorporating short-range wireless connectivity (e.g., Bluetooth [3]) and Internet connectivity (e.g., GPRS [8]) in the same personal mobile device.

Smart Phones can communicate with various embedded systems using short-range wireless networking. We define a protocol that allows a Smart Phone to query an embedded system about its identity and description. Using this information, the phone can download from the Internet (over the cellular link) an appropriate user interface for interaction with the embedded system. Additionally, the Smart Phone may download any data specific to the embedded system. Yet another benefit of the Internet connectivity of the mobile phone is a simpler security architecture that ensures authentication and access control during the interactions among Smart Phones and embedded systems.
2 Smart Phones Technology

With more than a billion mobile phones being carried around globally by consumers of all ages, the mobile phone has become the most pervasive pocket-carried device. We are beginning to see the introduction of Smart Phones, such as Sony Ericsson P800/P900 [7] and Motorola A760 [11], as a result of the convergence of mobile phones and PDAs (e.g., Figure 1). Unlike traditional mobile phones, which have limited processing power and act merely as “dumb” conduits for passing voice or data (Circuit Switched Data and Short Message Service) between the cellular network and end users, Smart Phones combine significant computing power with memory, short-range wireless interfaces (e.g., Bluetooth), and various input-output components (e.g., high-resolution color touch screens, digital cameras, and MP3 players).

Sony Ericsson P800/P900 runs Symbian OS [13], which is an operating system specifically designed for resource constrained devices such as mobile phones. It also comes equipped with two versions of Java technology: Personal Java [12] and J2ME(CLDC/MIDP) [1]. Additionally, it supports C++ which provides low level access to the operating system and the Bluetooth driver. Personal Java supports JNI which can be used to interface Java programs with C++ programs. The phone has 16MB of internal memory and up to 128MB external storage through the Memory Stick Duo Slot (i.e., external flash memory). There are many third-party vendors like Borland [4], CodeWarrior [5], AppForge [2], UIQ [15] which provide IDEs (Integrated Development Kits) to help developers accelerate the development process. In addition, this phone comes with pre-installed Java APIs like the Wireless Messaging API [16] and Java Phone API [9] which greatly enhance its programmability.

Motorola A760 uses a version of MontaVista Linux, Motorola’s i250 chip for communications, Intel’s 200 MHz PXA262 chip (based on the XScale PXA250) for computing with 256 MB RAM, and J2ME(CLDC/MIDP) [1] Java technology. We expect a wide range of development kits to be publicly available soon.

Bluetooth [3] is a standard for wireless connectivity based on a low-cost, low-power and short-range radio link. It uses radio transmission in the license-free band of 2.4 GHz and can transmit voice and data with bit rates up to 720 kbps. Originally developed as a cable-replacement technology, Bluetooth allows a collection of devices to form ad-hoc networks. Today, we can find Bluetooth technology in PCs, laptops, digital cameras, GPS devices, Smart Phones, and a whole range of other devices.
electronic devices. Bluetooth technology supports point-to-point and point-to-multipoint connections. We can actively connect a Bluetooth device to up to seven devices simultaneously. Together, they form an ad hoc network, called Piconet. Several piconets can be linked to form a Scatternet. Bluetooth has the capability to discover other Bluetooth devices in the proximity, which is a feature that can be exploited by proximity-aware applications.

Another important development for the mobile phone technology is the introduction of General Packet Radio Service (GPRS) [8], a packet switching technology over current GSM cellular networks. GPRS is offered as a non-voice value-added service that allows data to be sent and received across GSM cellular networks at a data rate of up to 171.2 kbps, and its goal is to supplement today’s Circuit Switched Data and Short Message Service. GPRS offers an always-on service (i.e., information can be sent or received when needed, subject to radio coverage) and supports Internet protocols.

We envision a wide spread use of Smart Phones as people upgrade their mobile phones over the next few years. The powerful combination of programmable ubiquitous Smart Phones, short-range wireless technology (Bluetooth), and connectivity to the Internet over GPRS is the cornerstone of our approach. In essence, the Smart Phones act as mobile gateways capable of connecting to the local embedded devices (supporting short-range connectivity) at one end, and the Internet at the other end.

3 Smart Phone Interaction Models

A Smart Phone can be used to interact with the surrounding environment in different ways. We have identified four interaction models: universal remote control, dual connectivity, gateway connectivity, and peer-to-peer. With these models, a Smart Phone can be used to execute applications from as simple as remotely adjusting various controls of home appliances or opening smart locks to complex applications such as automatically booking a cab, or ordering/paying in a restaurant using an ad hoc network of mobile phones to connect to the cashier’s computer.

3.1 Universal Remote Control Model

The Smart Phone can act as a universal remote control by performing interactions with embedded system located in its proximity. To support proximity-aware interactions, both the Smart Phone
and the embedded systems with which the user interacts must have short-range wireless communication capabilities. Figure 2 illustrates such interactions using Bluetooth. Due to its low-power, low-cost features, Bluetooth is the primary candidate for the short-range wireless technology that will enable proximity-aware communication.

Since multiple embedded systems with different functionalities can be scattered everywhere, the Smart Phone should be able to discover automatically or on-demand the embedded systems located in the proximity of the user. This is done using a short-range wireless device discovery protocol. At discovery time, the Smart Phone learns the identity and the description of these systems. Each embedded system should be able to provide its identity information (unique to a device or to a class of devices) and a description of its basic functionality in a human-understandable format.

This model works well as long as the user has pre-installed on the phone the interfaces for interacting with the embedded systems. An alternative and more flexible solution is to define a protocol that allows a Smart Phone to learn the interfaces from the embedded systems themselves. The problem with this idea is that many embedded systems may not be powerful enough to run complex software that implements such protocols. In the following, we describe a second model of interaction that offers a solution to this problem.

### 3.2 Dual Connectivity Model

Central to our proposal of building a universal interaction architecture based on Smart Phones is the dual connectivity model, in which the user connects both to the close-by environment and to the rest of the world through the Internet. This model of interaction, is based on the hybrid communication capabilities incorporated in these Smart Phones. They have the unique feature of incorporating short-range wireless connectivity (e.g., Bluetooth) and Internet connectivity (e.g., GPRS) in the same personal mobile device. This is the enabling feature for a secure and generic framework of services that we plan to develop over these phones. Figure 3 illustrates the Dual Connectivity interaction model.

As a typical application, let us assume that a person has just bought an “intelligent” microwave oven equipped with a Bluetooth interface. This embedded system is very simple and is not capable of storing or transferring its interface to the Smart Phone. However, it is able to identify itself to the Smart Phone. Using this information, the phone can connect to a server across the Internet (i.e., over GPRS) to download the code of the interface that will allow it to become a remote control for the microwave oven. 1. The phone can also perform authentication over the Internet to ensure that the code is trusted. All further communication between this embedded system and the Smart Phone happens by executing the downloaded code. This code will display on the phone’s screen a panel that emulates the panel of the microwave (i.e., it effectively transforms the phone into an intuitive microwave remote control). If we take a glimpse to the future, we expect to see recipes downloaded from the web that can be downloaded and executed on the microwave. For this purpose, the microwave does not have to be connected to the Internet; it is the smart phone that facilitates this action.

Another typical application is opening/closing Smart Locks. We envision that the entry in certain buildings will soon be protected using Smart Locks (e.g., locks that are Bluetooth-enabled and can be opened using digital door keys). The dual connectivity model enables users carrying Smart Phones to open these locks in a secure manner. The Smart Phone can establish a connection with the lock, obtain the ID of the lock, and connect to an Internet server over GPRS to download

---

1 This model of downloading the interface for a service or a device is similar to solutions proposed by Sun’s JINI [10] or HP’s CoolTown project [21]
the code that will be used for opening the lock (a digital door key can also be downloaded in the same time). The server hosting the interface and the keys for the Smart Lock maintains a list of people that are allowed to open the lock. The identity of the Smart Phone user (stored on the Smart Phone in the form of her personal information) is piggybacked on the request submitted to the server. If the server finds that this user is allowed allowed to open the lock, it responds with the code for the interface and the digital key.

The dual connectivity model can also be used to implement electronic payment applications similar to the Millicent [14] idea. A client does not need to know about a vendor’s embedded system in advance. The Smart Phone can authenticate the vendor using its Internet connection. The same connection can be used by the client to withdraw electronic currency from her bank and store them on the phone. Another option provided by the Smart Phone is to send some of the unused money back into the bank account (i.e., make a deposit each time the amount on the phone exceeds a certain limit). Potentially, the vendor’s embedded system can also be connected to the Internet. For instance, this ability can be used to authenticate the client. Figure 3, presents a similar application that involves accessing an ATM using a Smart Phone.

3.3 Gateway Connectivity Model

Many pervasive applications assume wireless communication through the IEEE 802.11 family of protocols. These protocols allow for a significant increase in the communication distance and bandwidth compared to Bluetooth. Using these protocols, the communication range is 250m or more, while Bluetooth reaches only 10m. The bandwidth is also larger, 11-54Mbps compared to less than 1Mbps for Bluetooth. Additionally, many routing protocols for mobile ad hoc networks based 802.11 already exist [23, 20]. The disadvantage of 802.11 is that it is expensive in terms of power consumption. With the current state of the art, we do not expect to have 802.11 network interfaces embedded in Smart Phones or other resource constrained embedded systems that run on batteries. More powerful systems, however, can take advantage of the 802.11 benefits and create mobile ad hoc networks. In such a situation, a user would like to access data and services provided by these networks from its Smart Phone. To succeed, a gateway device has to perform a change of protocol from Bluetooth to 802.11 and vice-versa.

Figure 4 illustrates this communication model and also presents an application that can be built on top of it. Let us assume a scenario where people want to book nearby cabs using their Smart Phones. Instead of calling a taxi company or ”gesturing” to book a cab, a client can start
an application on her Smart Phone that seamlessly achieves the same goal. Hence, the client is just one-click away from booking a cab. In this scenario, each cab is equipped with 802.11 wireless networking and GPS devices, and the entire booking process is completely decentralized. To join the mobile ad hoc network created by the cabs, a Smart Phone needs to connect to a gateway station that performs a translation of protocols from Bluetooth to 802.11 and vice-versa.

Many places in a city (e.g., stores, theaters, restaurants) will provide such gateway stations together with 802.11 hotspots. For instance, a Smart Phone can send a request for a free cab in the mobile ad hoc network formed by the cabs through such a gateway station. As soon as a cab has been booked, a message informing the client that a cab is on its way (as well as the license plate number of the cab) will be displayed on the phone’s screen. An underlying protocol takes care of all the details related to discovering and booking a free cab. Given that it is not yet feasible to have a GPS device incorporated in a mobile phone (mainly for power reasons) and that the current cellular location services are not very accurate, the gateway station can help the client communicate its location to the cab. The station can “stamp” the request for a free cab with its own street address (i.e., the client is nearby since she was able to connect using Bluetooth).

3.4 Peer-to-Peer Model

The Smart Phones can also communicate among them (or with other Bluetooth-enabled devices) in a multi-hop, peer-to-peer fashion, similar to mobile ad hoc networks. For instance, this model allows people to share music and pictures with others even if they are not in the proximity of each other. Figure 5 depicts yet another example of this model. A group of friends having dinner in a restaurant can use their Smart Phones to execute a program that shares the check. One phone initiates this process, an ad hoc network of Smart Phones is created, and finally the payment message arrives at the cashier.

4 Software Architecture

Figure 6 shows the software architecture of our system. This architecture applies to all of the four proposed interaction models. In the following, we briefly describe the components of this architecture.
• **Bluetooth Engine** is responsible for communicating with the Bluetooth-enabled embedded systems. It is composed of sub-components for device discovery and sending/receiving data. Although the Java API for accessing the Bluetooth stack has been proposed, it has not yet been implemented. *Bluetooth Engine* is a layer above the Bluetooth stack and provides a convenient Java API for accessing the Bluetooth stack. The downloaded interface is a Java program which cannot access the Bluetooth stack directly. It depends on the Java API provided by the Bluetooth Engine for communicating with the embedded device.

• **Internet Access Module** carries out the communication between the Smart Phone and various Internet servers. It provides a well-defined API that supports operations specific to our architecture (e.g., downloading an interface). The protocol of communication can be either HTTP or TCP/IP (on top of GPRS).

• **Proximity Engine** is responsible for discovering the embedded systems located within the Bluetooth communication range. If an interface for a newly encountered system is not available locally (i.e., a miss in the *Interface Cache*) or through direct communication with the system, the *Proximity Engine* invokes the *Internet Access Module* to connect to an Internet server and download the interface for interacting with the device. The downloaded interface is stored in the *Interface Cache* for later reuse. Together with the interface, an access control handler can also be downloaded. Such a handler executes before any subsequent executions of this interface. The *Proximity Engine* informs the *Execution Engine* to dispatch the downloaded interface for execution. All further communication between the Smart Phone and the embedded system happens as a result of executing this interface.

• **Execution Engine** is invoked by the Proximity Engine and is responsible for dispatching the downloaded interface program for execution. The downloaded interface interacts with the Bluetooth Engine to communicate with the embedded system or with other Smart Phones (as described in Section 3.4). This interface may also interact with the *Internet Access Module*.
to communicate with the webserver. For instance, it may need to contact the webserver for security-related actions or to download necessary data in case of a miss in the *Personal Data Storage*.

- **Interface Cache** stores the code of the downloaded interfaces. This cache avoids downloading an interface every time it is needed. An interface can be shared by an entire class of embedded systems (e.g., Smart Locks, or Microwaves). Associated with each interfaces is an access control handler that executes before any subsequent invocation of this interface (e.g., it checks if the interface is still allowed to run, sets the permissions to local resources).

- **Personal Data Storage** acts as a cache for “active data”, similar to Active Cache [18]. It stores data that needs to be used during the interactions with various embedded systems. Each data item stored in this cache has a number of handlers associated with it that can perform various actions (e.g., access handler, miss handler, eviction handler). Examples of such data include digital door keys or electronic cash. Each time an interface needs some data, it checks this cache. If the data is available locally (i.e., hit) the program goes ahead; otherwise (i.e., miss), it has to use the *Internet Access Module* to download the data from the corresponding server in the Internet.

We consider that any embedded system is registered with a trusted webserver (the webserver is just a simplification, since this is in fact a web service distributed on multiple computers). At registration, the webserver assigns a unique ID and a URL to the device. All the information necessary to interact with the device along with a user interface is stored at that URL. This URL may be common for an entire class of embedded systems.

Figure 7 shows the interaction protocol that takes place when a Smart Phone needs to interact with an embedded system. The user invokes the *Proximity Engine* each time she needs to interact with a device located in the proximity. Once the embedded systems in the proximity are identified, the user chooses the one she wants to interact with. A request is sent to the embedded system to provide its ID and URL. Upon receiving the ID and URL of the embedded system, a Smart Phone executes the access control handler and then loads and executes the interface. In case of a
miss in the *Interface Cache*, the interface needs to be downloaded on the phone either from the webserver or from the embedded system itself. An interface downloaded from an embedded system is un-trusted and is not allowed to access to local resources on (i.e., sandbox model of execution where it can only execute commands on the device). The interfaces downloaded from the webserver are trusted; they are assumed to be verified before being distributed by the server.

The Smart Phone requests an interface for the ID provided by the device (using the URL provided also by the device). With this request, the phone sends also its ID (stored in the *Personal Information Storage*). The Smart Phone is granted permission to download an interface, subject to the access control enforced based on the Smart Phone ID and, potentially, other credentials presented by the user. Once the access is granted, the webserver responds with the interface needed for any further interaction with the device.

The data stored in the *Personal Data Storage* can be classified into three categories:

- **Description of the downloaded interfaces.** The code of the downloaded interfaces is stored in the *Interface Cache*, but its description is stored in the *Personal Data Storage*. Every downloaded interface has an ID (which can be the ID of the embedded system or the class of embedded systems it is associated with). This ID helps in recognizing the cached interface each time it needs to be looked up in the cache. In addition to assigning an ID, the access and miss handlers associated with the interface are also defined. For instance, the user might define the time period for which the interface should be cached, or how and when it can be reused.

- **Confidential data.** Example of confidential data include electronic cash or digital keys. Every confidential data item has a data-ID and handlers associated with it. The handlers define a mechanism for evicting the data entity, sharing it across applications, or controlling the access to it. For instance, electronic cash can be sent back to the bank at eviction. The handlers also let the Smart Phone user have control over the confidential data stored on the phone. Any application that needs to access a confidential data item accesses it through the handler. The data item handler may either have an access control list of applications that are allowed to access it, or may pop up a choice menu to the user every time an application tries to access the data entity. Handlers are also provided for fetch data in from the corresponding server when it is missing in the cache.

- **Personal information of the user.** Personal information of the user in form of his name, credit card information, etc, is stored in this cache. This information is used for authenticating the user to applications that require that level of security, such as the digital-key application mentioned earlier. Personal information is primarily accessed by the architecture components of the Smart Phone. Applications may access this information through the handler associated with this data.

5 Related Work

Personal Server [24] is a mobile device of roughly the size of a packet of cigarettes, without standard physical I/O capabilities such as a keyboard, buttons or a display, and composed of three technologies: high-density, small volume storage; low-power, high-performance processors; and standardized low-power, high-bandwidth radios. The key observation for a Personal Server is the fact that the essence of a computer is the data stored in the hard disk. Rather than incorporating a small display
and minimal input device, mobile devices should instead rely on the computing infrastructure (i.e., keyboard and display) that happens to be in the vicinity and short-range wireless connectivity to make use of these resources. Effectively, a Personal Server virtualizes the hard disk through a wireless connection to whatever computing device is nearby and available.

CoolTown [21] proposes web presence as a basis for bridging the physical world with the World Wide Web. For example, entities in the physical world are embedded with URL-emitting devices (beacons) which advertise the URL for the corresponding entities. Our model is more general as we allow code and data download to the mobile device, either from the physical environment via short-range wireless connection, or from the Internet via the GPRS connection.

Microservers [19] share one of our goals of turning a mobile device (i.e., Smart Phone) into a universal remote control. Their approach relies on Web-based protocol (HTML and WAP) by embedding a web server in the device. Our approach is more general as we are not limited to Web-based protocol and use short-range connectivity as well.

Jini [10] is a system designed to deal with resource discovery in a new environment and interaction with the resources. When a service joins a network of Jini technology-enabled services, it advertises itself by publishing an object that implements the service API. The client finds services by looking for an object that supports the API. When it gets the service’s published object, it downloads any code it needs in order to talk to the service (via RMI, CORBA, XML, or any private protocols). Our approach encompasses both issue of resource discovery and interaction. In addition, we plan to develop a different security mechanism and semantic cache mechanism.

The idea of using digital keys to unlock doors has already been proposed [17] as a part of the Personal Server framework. Our work uses the Smart Phone as an incarnation of a Personal Server. However, the issue of key distribution from the external authority to the Personal Servers is not addressed. The digital keys work represents one of the many applications enabled by our proposed framework, which can distribute the keys over the GPRS connection.

References


