Accessing Ubiquitous Services Using Smart Phones

Nishkam Ravi, Peter Stern, Niket Desai, and Liviu Iftode
Department of Computer Science, Rutgers University
{nravi, peterst, nickd, iftode}@cs.rutgers.edu

September 8, 2004

Abstract

The integration of Bluetooth service discovery protocol (SDP), and GPRS internet connectivity into phones provides a simple yet powerful infrastructure for accessing services in nomadic environments. In this paper, we discuss the design and implementation of a protocol, called SDIPP, which unifies service discovery, interaction and payment through the use of Smart Phones. Although several service discovery protocols have been proposed earlier, such as SLP, Jini, UPnP, Salutation, they all have their own infrastructure requirements and target audiences. Bluetooth SDP is an on-the-fly service discovery protocol. However, it is not nearly as powerful as its counterparts. SDIPP works by augmenting Bluetooth SDP with web access and personalization. Payment of services has been overlooked in the protocols proposed earlier. SDIPP provides a novel protocol for payment via Smart Phones, based on Millicent scrips. We have implemented a few services to illustrate our protocol. We report on our experiences and experimental results. In particular, we analyse and provide an application level solution to the *Bluetooth inquiry clash* problem that we discovered in the process.

1 Introduction

Smart Phone technology is a result of the convergence of cell phones and PDAs and is steadily becoming ubiquitous with all the big mobile manufacturers like Nokia, Sony Ericsson and Motorola vigorously supporting it. Despite efforts by giants like Miscrosoft and Palm, and presence of Linux based Smart Phones like Motorola A760, Symbian OS remains the most dominant platform for Smart Phones, holding 67% of the market share which is expected to grow. Symbian OS supports Personal Java, J2ME, MIDP as well as C++. With ample memory and computing power, Smart Phones are expected to personify the *abstract handheld* that has so long been part of the picturesque visions of pervasive computing [30]. Many believe that Smart Phone technology will be instrumental in guiding and realising the visions of pervasive computing, by giving shape to abstract ideas and assumptions.

Several service discovery protocols have been proposed, such as SLP [21], Jini [7], UPnP [8], Salutation [12], Bluetooth [13], DEAPspace [24], Intentional Naming System [15], Secure Service Discovery Service [19] and Splendor [31]. These protocols can be roughly classified into two categories: *client-service* model and *client-service-directory* model, with the exception of Splendor which follows a *client-service-proxy-directory* model, where *proxy* is used to achieve privacy, authentication and loadsharing. While the *client-service* model requires less infrastructure and is more suitable for nomadic environments, it is not nearly as powerful as the *client-service-directory* model. For the widespread deployment of one or more of these protocols, clients and directories should take a form acceptable by end-users. We take the stand that Smart Phones are suitable candidates for clients and build our solution around this assumption.

Bluetooth SDP has been incorporated in Smart Phones and follows the *client-service* model. However, it can be augmented to support the *client-service-directory* model by using the GPRS connectivity on the phones without the need of any additional infrastructure. Hence, in our solution directories take the form of web servers. Directory lookup has additional cost associated with it (in terms of both time and cost of downloading data from internet) and its usage should be minimized.

Bluetooth SDP leaves open the interaction between clients and services after discovery. Interaction between clients and services involves obtaining the interface for communicating with the service and in the case where the service is profit-based, also involves payment for service usage. Payment for service usage should not be a part of the service interaction protocol as the services are assumed to be untrusted. The service discovery protocols proposed earlier overlook the payment aspect of service usage.

A person driving down a street with several restaurants by the side may want to order food from her phone without having to obtain numbers and calling up every restaurant. In addition she may want to schedule a pickup or delivery. In the pervasive computing world, the restaurants may even install devices for receiving orders and taking payments at different locations. In such a scenario the user would do an *inquiry* for *food services*, and obtain a list of services that match her preferences stored on the handheld. She would select a service and automatically obtain, on her handheld, the interface for communicating with the service. After placing the order and specifying pickup/delivery details, she must pay for the service. She does not wish to use her credit card for this purpose and would rather use some form of anonymous payment system for privacy reasons.

Similarly, a person lost on a highway may want to avail of a service that provides her (prints or dictates) directions to help find her way. The service has been installed by a service provider at different locations on the highway and is profit-based.

The services mentioned above are examples of *nomadic profit-based services* and would be different in nature from *non-profit services* or *personal area services* in many ways, such as motivation for their existence, economic models driving them, access and discovery mechanisms, composition and description. *Profit-based* services would not only need strong revenue models in order for them to exist, but also widely accepted and simple to use access mechanisms. While secure payment and dynamic interaction would be issues, composition and interoperability would take a backseat. At the same time, assumptions about existence of any supportive infrastructure for discovery would be unrealistic.

With that in mind, we propose a minimal infrastructure and simple to deploy protocol for access. In the scenarios described above, there are three phases: service discovery, service interaction and service payment. We have designed a protocol called *SDIPP* (Service Discovery, Interaction and Payment Protocol) for Smart Phones that unifies the three phases. We have implemented and tested this protocol on Sony Ericsson P900 phones. SDIPP makes use of the personal information of the user stored on the phone to select suitable services for the user and also uses the location information for discovering services. We have implemented a few applications and evaluated them.

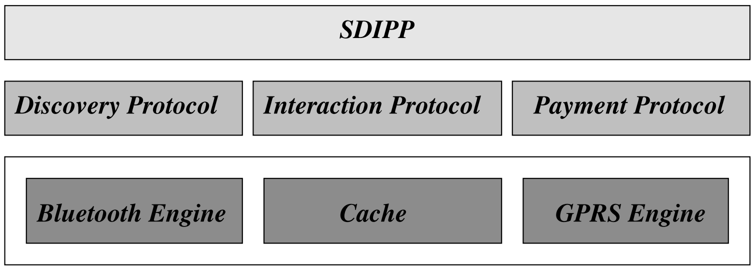


Figure 1: SDIPP Architecture

In the process, we have learnt lessons about Bluetooth on Smart Phones and we discuss them in the paper. In particular, we analyse and provide an application level solution to the *Bluetooth inquiry clash* problem that we discovered while implementing the applications.

In Section 2 we present the SDIPP protocol. In Section 3 we describe the applications we have implemented and discuss problems and solutions therein. In Section 4 we discuss the Bluetooth inquiry clash problem. In Section 5 we discuss related work. We conclude in Section 5 with directions for future work.

2 Service Discovery, Interaction and Payment Protocol (SDIPP)

If Smart Phones are fixed as clients, what is the best we can obtain. We try to answer this question while designing SDIPP. From the user's perspective, SDIPP is composed of three phases. In the first phase the services are discovered. In the second phase, mechanism for interacting with the service is figured out. In the third and final phase the user interacts with the service and pays for its usage.

2.1 Architecture

Figure 1 shows the SDIPP architecture. *Bluetooth engine, GPRS Engine* and *Cache* are the building blocks of the protocols. Bluetooth Engine is invoked by the protocols to discover or interact with the services in the proximity. It is a layer above the Bluetooth stack and provides a convenient Java API similar to JSR-82 for accessing the Bluetooth stack. *GPRS Engine* is invoked to carry out the communication between the phone and the web services over GPRS.

Cache is persistent storage. The personal information of the user along with her preferences regarding services are stored in the cache. Personal information of the user may include name, age, address, credit card number etc. Storing personal information serves two purposes: first, it provides a way of identifying the user and authenticating her if need be; second, personal information along with preferences and location help in identifying the best suited service for the user during service discovery phase thereby making SDIPP context-aware. Cache also stores the interface/protocol downloaded by the interaction protocol and the data needed across protocols or across sessions.

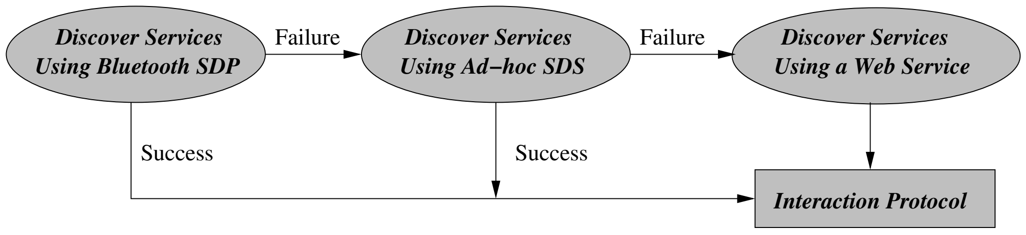

Discover Services Using Bluetooth SDP
Failure
Discover Services Using Ad−hoc SDS
Failure
Discover Services Using a Web Service
Success
Success
Interaction Protocol

Figure 2: Discovery Model

2.2 Discovery protocol

Our discovery protocol is heirarchical in nature and builds on top of Bluetooth service discovery protocol [13] which is *query based* and has already been incorporated in Smart Phones. Bluetooth SDP provides service browsing without apriori knowledge of the service characteristics. It does not include functionality for accessing services, however, it can be used in conjunction with any other protocol for accessing services. The discovery protocol is a 3-step process, as summarized in what follows.

One-hop discovery: Services in the proximity (one-hop) are discovered using Bluetooth SDP. If the list of services discovered by Bluetooth SDP includes the desired service, the discovery phase is over. If it does not include the desired service, but instead lists a Service Discovery Service(SDS), the SDS is invoked to locate the desired service in a multi-hop fashion.

Multi-hop discovery: SDS would implement a mechanism for discovering services and could exploit the ad-hoc network for doing so. SDS could be implemented using DEAPspace, SLP, Salutation, UPnP, Secure SDS [19], or Mobile Agents [28] [29] [18]. Effort has been made to map Salutation APIs to Bluetooth Service Discovery Layer [20]. We believe that such mappings would exist between other service discovery protocols and Bluetooth SDP. The other option is to have an ontology based service description layer with bindings to different service discovery protocols.

In an earlier work, we had ported Smart Messages on Smart Phones [27]. Smart Messages are user-defined distributed applications similar to mobile agents that execute on nodes of interest defined by properties. Smart Messages migrate between nodes of interest using content-based routing where content/property is stored in *TagSpace*. *TagSpace* is name-based memory and is composed of *tags*. A *tag* is a $(name, data)$ pair. We use Smart Messages for discovering services in an ad-hoc network. A service is identified by a tag that it creates on the device it runs on, for it to be discoverable by a Smart Message. The tag stores the service description. An SDS implemented using Smart Messages eliminates the need of having directories in the ad-hoc network. If a directory exists it can be exploited but the discovery does not depend on the presence of a directory.

Web-based discovery: If no SDS is listed, the GPRS connectivity on the phone is used to contact a web service that can locate the desired service. The services would have themselves registered on the web server as a way of advertising themselves and would periodically update their information on the web server. This web service would serve as a directory that lists the location of all the registered services around a particular location. We use a simple interval tree representation for storing location and services, however, a better representation would be ontological, which is out of the scope of this paper.

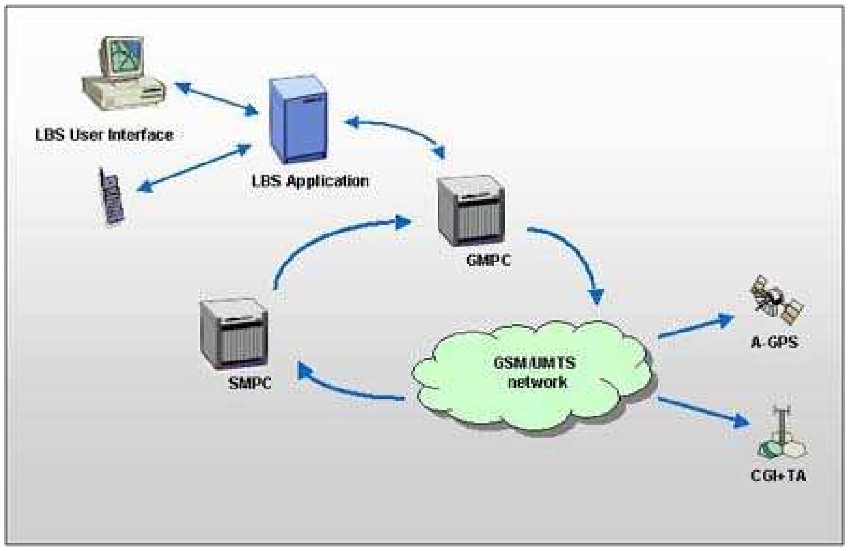
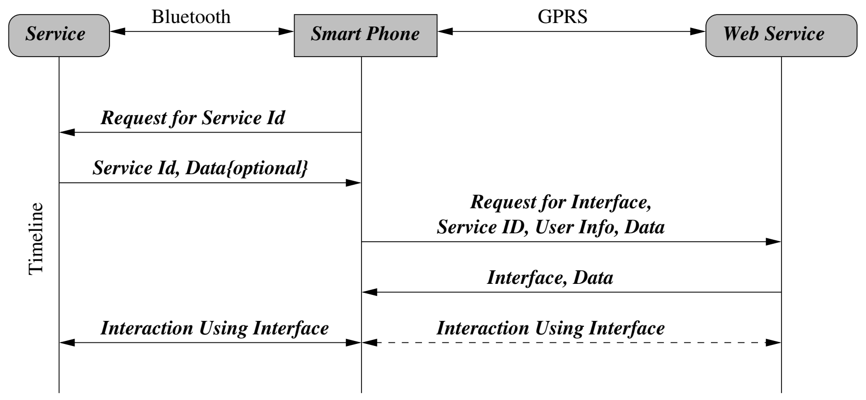


Figure 3: Mobile Positioning System

Since downloading data from the internet costs money, directory lookup is the last step in our protocol. The user stores her personal information and preferences in the *Cache* on the Smart Phone. The web service lists the services based on user's location, personal information and preferences. Figure 2 summarizes the discovery phase.

2.2.1 Determining Location

The web service that acts as the directory for looking up services, provides information based on the location of the client. There are several mechanisms for determining the location of a Smart Phone. Amongst the popular ones are Bluetooth localization [14] and GPS. In our implementation, the web directory discovers the location of the user using the Mobile Positioning System (MPS) recently released by Ericsson. Figure 3 shows the various components of MPS. MPS consists of a server-based Gateway Mobile Positioning Centre (GMPC), a server-based Serving Mobile Positioning Centre (SMPC) and software extensions for the operator's mobile network. The web directory is implemented as an LBS (Location Based Service) Application, which can request the GMPC for the position of the mobile phone. After performing an authorization, the GMPC forwards the request to SMPC via the GSM network. SMPC collects position information from the GSM network, converts it to the global coordinate format and delivers the coordinates to GMPC which forwards it to the LBS. Thus by just knowing the MSISDN of the phone (which is passed on as a parameter in the http request), the LBS (web directory) can determine the location of the phone. MPS supports the standardized Mobile Location Protocol (MLP), which is the interface between the LBS and GMPC. MPS that uses the Assisted Global Positioning System (A-GPS) gives a high accuracy (within 10m radius). For details refer to [4]. Since MPS is a service provided by the cell phone service providers, using it avoids the need of using an external device like GPS or an algorithm like Bluetooth localization (which requires infrastructure) for determining location of the phone.



Bluetooth
GPRS
Service
Smart Phone
Web Service
Timeline
Request for Service Id
Service Id, Data{optional}
Request for Interface, Service ID, User Info, Data
Interface, Data
Interaction Using Interface
Interaction Using Interface

Figure 4: Interaction Protocol

2.3 Interaction protocol

In a ubiquitous computing environment, the interactions of the end-user with a service are assumed to be spontaneous, in which case the mechanism or protocol for interacting with the service would need to be learnt on the fly. Our interaction protocol is inspired by Jini [7]. Every service registers itself with a web server which assigns it a unique id and stores the interface which can be downloaded for interacting with the service. Figure 4 gives a pictorial view of the interaction protocol. The protocol can be summarized as follows:

- The Smart Phone lists the services discovered during discovery phase. A request is sent to the desired service to send back its unique id over the Bluetooth connection.
- The service responds with its id.
- The id along with the personal information of the user stored on the Smart Phone is sent over to a trusted web server over the GPRS connection. The personal information of the user would be used for authenticating her if the service requires that.
- The web server after an optional authentication responds with the code and data that will be used for interacting with the service. The code is a Java program which contains the protocol and interface for interacting with the service.
- Since the code is obtained from a trusted server it is assumed to be safe code and is dispatched for execution on the Smart Phone. All further communication between the phone and the service takes place as a result of executing the downloaded code.

Note that the web server(s) for storing the downloadable interface for the services may be different from the web server(s) that act as service directories. We implement downloading of code from the internet using OTA (Over-The-Air) provisioning [9] supported by most cell phone service providers for MIDP.

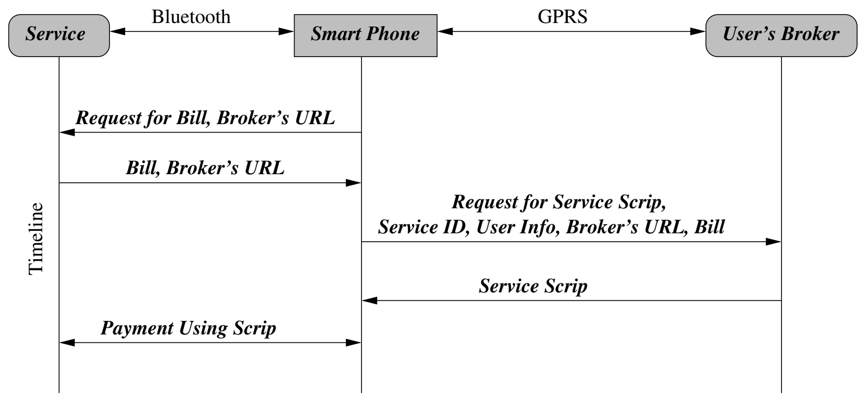


Figure 5: Payment Protocol

2.4 Payment protocol

We have designed a payment protocol for paying services using Smart Phones in nomadic computing environments, based on the electronic cash representation proposed by the Millicent protocol [10]. There are a number of existing and proposed protocols for electronic commerce such as DigiCash [3], CyberCash [2], First Virtual [5], NetBill [16] and Millicent [10]. None of these protocols has taken off due to lack of supporting infrastructure to implement them in real life.

Digital cash is normally issued by a central trusted entity (like a bank). The integrity of digital cash is guaranteed by the digital signature of the issuer, so that counterfeiting digital cash is extremely hard. However, it is trivial to duplicate the bit pattern of the digital cash to produce and spend identical (and equally authentic) cash.

Millicent proposes the idea of using accounts based on scrip and brokers to sell scrip. A piece of scrip represents an account the user has established with a vendor. At any given time, a vendor has outstanding scrip (open accounts) with the recently active users. The balance of the account is kept as the value of the scrip. When the customer makes a purchase with scrip, the cost of the purchase is deducted from the scrip's value and new scrip (with the new value/account balance) is returned as change. When the user has completed a series of transactions, he can "cash in" the remaining value of the scrip (close the account).

Brokers serve as accounting intermediaries between users and vendors. Customers enter into long-term relationships with brokers, in much the same way as they would enter into an agreement with a bank, credit card company, or Internet service provider. Brokers buy and sell vendor scrip as a service to users and vendors. Broker scrip serves as a common currency for customers to use when buying vendor scrip, and for vendors to give as a refund for unspent scrip.

We try to satisfy the design principals described in [25]. In our model, the broker is a web service that the user already has an account with. The vendor is the service that the user wishes to use and pay for.

However, Millicent model assumes that before the customer initiates transaction with the vendor, she either already has the vendor scrip or has the broker scrip which can be used to buy vendor scrip directly from the vendor. This is not a reasonable assumption to make in ubiquitous computing scenarios where transactions are primarily spontaneous. Having service/vendor scrip prior to discovering the service would imply that the user already had some knowledge about the service. This is clearly not acceptable. Having broker scrip prior to discovering the service is a reasonable assumption to make, however, in order to be able to use the broker scrip, the service in question should be able to verify the authenticity of the broker scrip which requires additional infrastructure.

Dual connectivity on the phones allows us to get rid of both these assumptions because the user can be connected to the broker and the service at the same time.

Figure 5 illustrates the payment protocol. It can be described as composed of the following steps:

- The Smart Phone requests the service for its broker's URL and the bill over Bluetooth connection.
- The service responds with its broker's URL and the bill.
- The service id, broker's URL and bill amount is sent over to the user's broker over GPRS along with the personal information of the user stored on the phone.
- User's broker buys service scrip from service's broker on user's behalf. The amount of scrip bought is greater than or equal to the bill amount.
- User's broker responds to the Smart Phone with the service scrip .
- User pays the service using the service scrip.

Brokers are assumed to be trusted services that have service providers as their clients and other brokers as their peers. Even if the broker tries to cheat, the customer and the service provider can independently check the scrip and detect broker fraud.

Service provider fraud consists of not providing service for valid scrip or deducting more amount from the scrip than is valid. If the service provider tries to cheat, the customer can detect the fraud and complain to the broker who will take care of it.

If the customer is cheating, then the service provider's only loss is the cost of detecting the bad scrip and denying service. Every transaction requires that the customer knows the secret associated with the scrip. The protocol never sends the secret in the clear, so there is no risk due to eavesdropping. No piece of scrip can be reused, so a replay attack will fail. Each request is signed with the secret, so there is no way to intercept scrip and use the scrip to make a different request.

This payment protocol provides a security model that is well suited for profit-based services where the service and the user need to be authenticated to each other and anonymity maintained at the same time.

2.5 Bootstrapping

The service registers itself on a web directory (for discovery) and a web broker (for payment). In addition the service uploads the interface and data needed to interact with it on a web service after authentication and certification. For it to be discoverable by an SDS implemented using Smart Messages, the service creates a *tag* containing the service description.

Table 1: Performance Evaluation

Operation	Average Time of Completion
Bluetooth Service Discovery	22.5 sec
Ad-hoc Service Discovery	8 sec $\times$ No. of Hops
Web directory lookup	2.5 sec
Interaction Protocol(Lower Bound)	3 sec
Payment Protocol	6 sec

3 Applications and Evaluation

The SDIPP protocol was implemented and tested on Sony Ericsson P900 phones which have both Personal Java and MIDP in addition to C++. We used MIDP and JSR-82 (Java Bluetooth API) to implement the architecture. Table 1 shows the time of completion for the different phases of the SDIPP protocol. The time of completion of the Interaction Protocol depends on the size of the code downloaded from the internet. The lower bound is determined by the size of the *jad* file of the corresponding code which is typically 250 Bytes. The time of completion of the ad-hoc service discovery over Smart Messages depends on the number of nodes (hops) involved.

Smart Messages(SM) were ported and tested on Sony Ericsson P800 and P900 phones. Personal Java and C++ (connected through JNI) were used to port SMs on phones. We realized that due to limited computing power on the phones, the context switching overhead is quite high and performance falls noticeably for multithreaded programs. While the Round Trip Time of an SM on HP iPAQs communicating via 802.11b is around 125 milliseconds, the Round Trip Time of the same SM on P800 phones communicating via Bluetooth L2CAP is around 16 seconds. This is due to limited computing power and limited bandwidth.

Device and service discovery together on an average takes around 22.5 seconds to complete on P900 phones when there are 3-4 Bluetooth devices around. We noticed irregularity in the behavior of Bluetooth protocol. At least once in every 20 runs, service discovery would fail. Similarly, if a phone has been using Bluetooth radio for a long time, its Bluetooth performance falls and service discovery fails repeatedly (the phone was continually on charge so battery was not the problem). Rebooting the phone solves the problem. This could be due to crowding of service records in the service record database on the phone. But we could not draw any definitive conclusions. The same problems were exhibited while using C++ instead of JSR-82.

We noticed that when two phones are simultaneously in inquiry mode, they become undiscoverable to each other which is due to the fact that, when in inquiry mode the Bluetooth radio bandwidth is completely consumed. We give an analysis of this problem in detail in the next section and offer a solution at the application level. We have implemented two applications to evaluate the feasibility of SDIPP protocol. In this section we briefly describe the applications.

3.1 Restaurant Application

This application was motivated by the scenario mentioned earlier in the paper where a person driving down a road with restaurants by the side wants to order food and pay the bill using his Smart Phone. The service runs on a bluetooth enabled device installed by the restaurant and can be discovered and interacted with using the SDIPP architecture. We simulate the device by a bluetooth enabled computer. The service registers itself on a web directory (for discovery) and a web broker (for payment). In addition the service uploads the interface and data needed to interact with it on a web service after authentication and certification. For it to be discoverable by an SDS implemented using Smart Messages, the service creates a *tag* containing the service description.

The user's phone discovers the services using the discovery protocol. The discovery protocol lists the services that match user's preferences. User's preferences are specified as short strings, such as *cheap food* or *mexican food* under the category of *food preferences* on his Smart Phone and this is matched with the service profile obtained by the discovery protocol while discovering the services. This could be done in a more effective and generic way by using ontologies [26], however, we use our own representation of service profiles for the purpose of this application.

Once the service is selected, the interface and data for interacting with it is downloaded from the internet over the GPRS connection. The code is dispatched for execution which fetches the menu and allows the user to place his order. After the user has placed the order, payment is made via the payment protocol.

Thus, all the user needs, to avail of such a service is the SDIPP architecture on his phone. Payment protocol takes, on an average, around 6 seconds to complete. The interaction protocol (downloading code and data from the internet) takes around 5 seconds to complete. For this application 9Kb of data was downloaded from the internet using T-Mobil as the service provider. Discovery time dominates and varies from 22.5 seconds to a minute depending on which mechanism was finally effective in discovering the services. This application was tested on Sony Ericsson P900 phones.

The same application can be used in the context of a user dining with her friends in a restaurant. We have implemented an extension of this application that lets the user split the restuarent bill with her friends. The bill migrates between the phones of the friends (selected by the user) and everyone is asked to contribute some amount which is then accepted or rejected by the initiator. The full bill is payed by the initiator but IOU's are maintained on the phones. The bill splitting part as of now does not support any authentication.

3.2 Door Application

Beaufour et al [17] had proposed the idea of opening doors using digital keys. However, they assume that the keys already exist on the phone which limits the power of the application. We have implemented this application on top of the SDIPP architecture on P900 phones and demonstrated it on the doors of the Computer Science Department.

Our implementation allows the user to download the keys from a trusted webserver after the user authenticates himself on the web server using his personal information that uniquely identifies him (a combination of social security number and credit card number for example). The user has to be registered with the web server which maintains an access control list and key database for

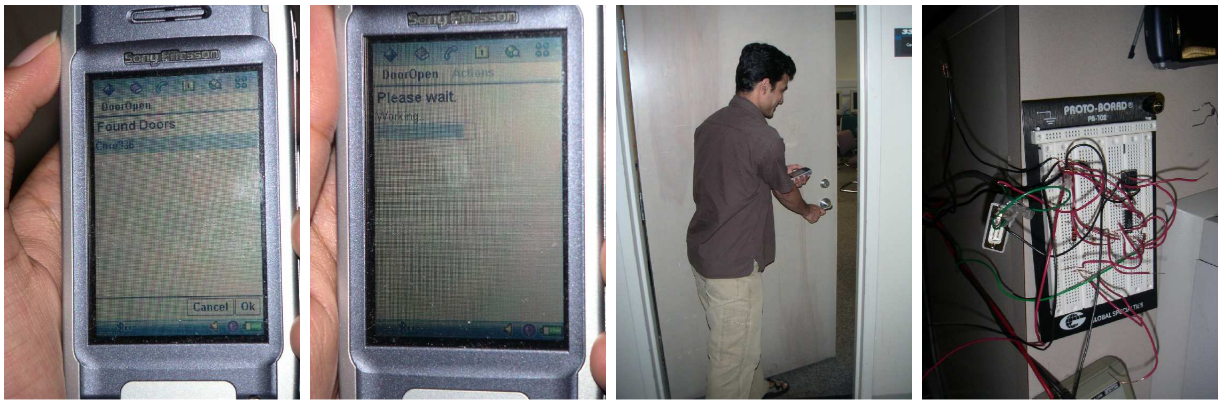


Figure 6: Door Application

every digital door.

After discovering the digital doors around him using the discovery protocol, the user's phone downloads the interface and key for opening the door from the web server. Personal information of the user is piggybacked on the request to the web server. The web server releases the key after authenticating the user on the basis of his personal information. The phone then interacts with the door device by executing the downloaded interface. Key once downloaded can be reused by the user for future use.

We would like to incorporate the feature of sharing keys with friends, but there are security issues surrounding it. The ideal solution would be to let the owner of the digital key replicate or derive a key that can be transferred to a friend's phone in such a way that the derived key can neither be tampered with nor replicated by the friend. We do not have a clear algorithm for implementing this solution yet and hence we have not yet implemented the key sharing feature.

We have demonstrated this application on the doors of our department. We simulated the Bluetooth enabled door device by a Smart Phone which runs the *Door Open* service. The user's phone only interacts with the *Door Open* service and the web server that stores the keys. The door device communicates with a backend computer that actually controls the doors.

To make it real, we built our own control circuit driven by the backend computer's serial port that controls the power supply to the door lock. On receiving a signal from the door device, the computer sets the Data Terminal Ready Bit to 1 which switches on a 24V power supply to the door lock thereby opening it. Excluding discovery, the whole procedure takes around 5.5 sec to complete. Since there is no payment involved, the payment protocol is never invoked in this application.

4 Bluetooth Inquiry Clash

When a Bluetooth device is in the inquiry mode, it continuously transmits inquiry packets and hops frequencies 3200 times per second. A device that allows itself to be discovered, regularly enters the inquiry scan state to respond to inquiry messages, hopping frequencies once every 1.28 seconds. In order to discover all devices, the device must spend at least 10.24 seconds in the inquiry mode [14] [13]. Once discovery has completed, the device behaves like other devices, entering the inquiry scan state periodically. When device discovery is followed by service discovery the minimum time to be spent in the inquiry mode is even larger. We have observed that on an average a P900 phone takes 22.5 seconds to discover all devices and services. Inquiry is a process that inside most basebands takes up all the radio bandwidth and we have observed this to be true for P900 phones. This means that a phone, when in the inquiry mode, cannot use the radio for anything else except actions like *local friendly name retrieval* or *local parameter retrieval*, which implies that the services on the phone become undiscoverable while it is in the discovery mode.

In order to analyse and demonstrate the effect of this problem we implemented the *Profile Exchange* service. This is an application to discover and retrieve the Profiles/Contact Information stored on other people's Smart Phones. This application was implemented in Java MIDP and tested on P900 phones. In order for a remote profile to be discoverable, the owner of that phone must have the phone in discoverable mode and the Profile Exchange service must be running in the background. The phone owner can explicitly search for profiles or choose the *polling mode* in which case the phone polls for available profiles periodically and notifies the user with a beep whenever a new profile is discovered. Optionally the user can do a selective profile search by specifying a keyword (filter). When this option is selected, all subsequent inquiries will only retrieve profiles that contain the keyword. This option is helpful when a traveller on a train is looking for a *doctor*. In general, the application is helpful in meetings and conferences where people want to know each other and exchange contact information.

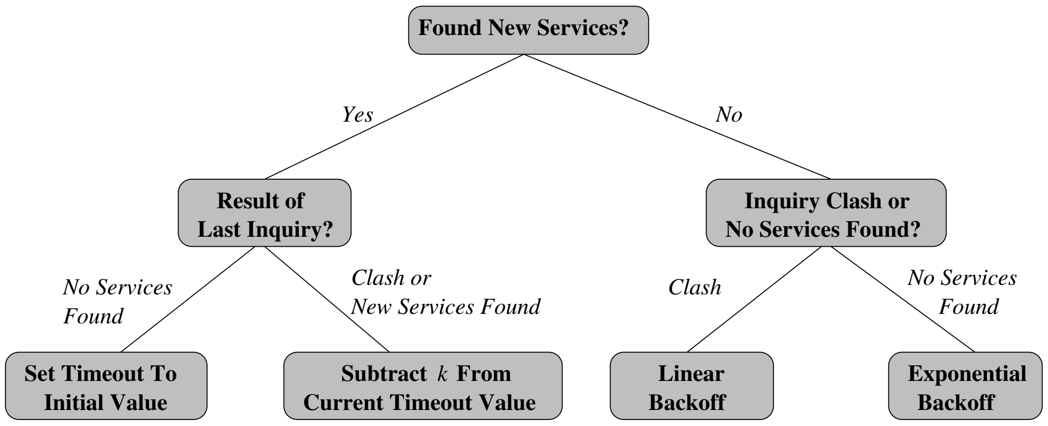
However, this application is effective only when there are many people running the service on their phones, each willing to exchange personal profiles. This implies that all those phones would be simultaneously in the polling mode trying to discover services on other phones, since polling is the only mechanism for discovery. As everyone would like to discover profiles as quickly as possible, the application would exhibit *greedy* behavior. This would result in inquiry clashes as mentioned above.

We can envision many such applications being deployed in the future where Bluetooth devices are continuously trying to discover other Bluetooth devices/services around them. In the *Door Application*, for example, the door devices could poll instead of the phones as in [17]. Similarly, in the *Restaurant Application* the restaurant devices might poll in order to push advertisements to the handhelds. Another example is that of the WAP-push based advertising system described in [14].

This problem is similar in nature to that of nodes connected on an ethernet, each trying to send data periodically, but unlike ethernet which is a static system, this is more dynamic in nature as new devices are encountered and old ones disappear due to mobility. The solution to this problem (at the application layer) lies in choosing the optimal timeout value t between two consecutive polls. A very small t would result in large number of clashes while a very large t would result in inefficiency.

If the timeout t is constant, two phones running the same application and hence polling after every t seconds will never be able to discover each other. A better choice would be to choose t randomly between a certain interval $[0, r]$. Let c denote the time it takes for discovery to complete, during which time the services on the device become undiscoverable. Every device is discoverable with probability $1 - c/r$.

Let X_i be a random variable denoting the time instant when device i started discovery, $0 \leq$



Found New Services?
Yes
No
Result of Last Inquiry?
Inquiry Clash or No Services Found?
No Services Found
Clash or New Services Found
Clash
No Services Found
Set Timeout To Initial Value
Subtract k From Current Timeout Value
Linear Backoff
Exponential Backoff

Figure 7: Policy for Timeout

Table 2: Comparing *Random* With Our Policy

Number of Phones	Discovery Completion Time	
	Random	Policy
3	10 min	5 min
4	18 min	12 min

$X_i \leq r$. The probability p of two devices i and j being able to discover each other in the time interval $[0, r]$ is given by

$$P(\mid X_i - X_j \mid > c) \tag{1}$$

which comes to $(1 - c/r)^2$. Let $Y_{ij}(i \neq j, Y_{ij} = Y_{ji})$ denote the number of attempts needed for i and j to discover each other. Y_{ij} follows a geometric probability distribution. Let Y be a random variable denoting the number of attempts after which n devices would have all discovered each other.

$$Y = \sum_{i,\, j>i} Y_{ij} \tag{2}$$

and therefore follows a negative binomial distribution. Expected value of Y is given by

$$E(Y) = \frac{\binom{n}{2}}{p} \tag{3}$$

where p is $P(\mid X_i - X_j \mid > c) = (1 - c/r)^2$

Let T be the random variable that denotes the time after which n devices would have all discovered each other.

$$T = rY \tag{4}$$

$$E(T) = rE(Y) = \frac{r\binom{n}{2}}{p} \tag{5}$$

In our case c is 22.5 seconds. We chose r to be $2c$ which is 45 seconds, for best results. Substituting the values, we obtain the expected time for $n = 3$ devices to be 9 minutes and for $n = 4$, it is 18 minutes. This matched very well with our experiments. However, this is far away from the ideal, which for 3 devices is $3c$ which is 67.5 seconds (1 minute 7.5 seconds) and for 4 devices, 90 seconds (1 minute 30 seconds). However, the ideal is achievable only with an oracle aware of all devices, that instructs the devices when to start discovery.

After careful analysis and experimentation we came up with a policy for varying t dynamically that seems to do much better than *random*. The policy was motivated by the backoff algorithms used to minimize clashes in an ethernet. In addition, it also takes into account the dynamic nature of the problem by keeping track of events that happened during the last discovery attempt. The user may suddenly enter a region where there are many new services available in which case he should poll more often, however, if there are no new services found or if a clash happens, then the value of t should be increased. Figure 7 shows the policy. For best results, original(initial) value of timeout t was chosen to be r (45 seconds) and k which is a constant, was chosen to be 15 seconds.

Table 2 shows how this policy compares with the *random*. We believe there is room for improvement. Better results could be obtained by using machine learning techniques to learn the policy for varying t.

5 Related Work

Aalto et al [14] describe a system for Bluetooth and WAP Push based advertising for Smart Phones. I-mode [6] makes web service provisioning on Smart Phones easier. UDDI [11] is a web-based distributed directory that enables profit-based web services to list themselves on the internet and discover each other, similar to *yellow pages*. Beaufour et al [17] propose the idea of storing digital keys on Smart Phones for opening doors. Jini [7] proposes the idea of downloading the published objects for interacting with the service, from other services. In an earlier work [22], we had published how Smart Phones can be used to support several ubiquitous computing models. Cooltown [23] and Splendor [31] utilize the idea of associating devices and services with the web. SDIPP goes a step further by utilizing web connectivity on phones for *(1).* augmenting a *service-client* based discovery protocol (Bluetooth SDP) *(2).* dynamically obtaining the protocol to interact with a service and *(3).* paying services in a seamless way. SDIPP also integrates ad-hoc service discovery using Smart Messages [27] with the web assisted Bluetooth SDP in a cost-sensitive heirarchical way and utilizes personal information, preferences and location information for discovery. SDIPP unifies several ideas to implement an infrastructure-less solution suitable for nomadic computing and has been shown to be effective in many scenarios described in the paper.

6 Conclusions and Future Work

In this paper, we have outlined the design of an infrastructure that provides a real-life solution to the problem of interacting with services in nomadic ubiquitous computing environments, by advocating the use of Smart Phones as clients and exploiting whatever resources are available on phones. In particular, we have sketched a unified protocol, SDIPP, for discovering, accessing and paying services dynamically using Smart Phones. This work is complementary to the work being done in designing service discovery protocols and payment systems for ubiquitous computing, which can be integrated with the framework described in the paper.

We have advocated the use of *dual connectivity* on phones for integrating local services with web services in a secure way. Context and location awareness are integrated into SDIPP. Although personal information is used for authentication, it is shared with only trusted web servers, without compromising privacy in a big way.

We have evaluated our protocol and demonstrated its feasibility through two applications. We pointed out the limitations of Bluetooth on Smart Phones and provided an application level solution for inquiry clash.

There are two major issues that we have not addressed in this paper: security and energy efficiency. Since personal information of the owner is stored on the phone, loosing it could pose a serious security threat to the owner. The data stored on the phone should be made inaccessible to anyone but the phone owner. A simple password scheme is insufficient because entering a password every time confidential data is accessed could be a major turn off for the users. We plan to investigate both software protection mechanisms and hardware solutions (e.g., biometric security using fingerprint or voice recognition [1]).

So far, there is no systematic characterization of the energy consumption on Smart Phones. Energy consumption is going to be a deciding factor in the success of Smart Phones. Since phones, as of now, are closed platforms, we plan to look into source code transformations for energy efficiency.

We have concentrated on Bluetooth services for the purpose of this paper. We plan to use ontology based service descriptions and implement groundings to Bluetooth and UPnP.

References

[1] HP iPAQ 5400. http://welcome.hp.com/country/us/en/prodserv/handheld.html.

[2] CyberCash. http://www.cybercash.com.

[3] Digicash. http://www.digicash.com.

[4] Ericsson Mobile Positioning System. http://www.ericsson.com/mobilityworld/sub/articles/.

[5] First Virtual Holdings Incorporated. http://www.fv.com.

[6] i-mode. http://www.nttdocomo.com/corebiz/imode.

[7] Jini Network Technology. http://wwws.sun.com/software/jini.

[8] Microsoft Upnp Specification.

[9] OTA Provisioning. http://java.sun.com/products/midp/OTAProvisioning-1.0.pdf.

[10] The Millicent Protocol for Inexpensive Electronic Commerce. http://www.w3.org/Conferences/WWW4/Papers/246/.

[11] UDDI. http://www.uddi.org.

[12] White Paper : Salutation Architecture, 1998. http://www.salutation.org/whitepaper/originalwp.pdf.

[13] Bluetooth Specification Part E. Service Discovery Protocol (SDP), 1999. http://www.bluetooth.com.

[14] L. Aalto, N. Gothlin, J. Korhonen, and T. Ojala. Bluetooth and wap push based location-aware mobile advertising system. In *Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 49–58. ACM Press, 2004.

[15] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The design and implementation of an intentional naming system. In *Proceedings of the seventeenth ACM symposium on Operating systems principles*, pages 186–201. ACM Press, 1999.

[16] B. Cox and J.D. Tygar and M. Sirbu. Netbill security and transaction protocol. In *Proceedings of the First USENIX Workshop on Electronic Commerce*, 1995.

[17] A. Beaufour and P. Bonnet. Personal Servers as Digital Keys. In *Second International Conference on Pervasive Computing and Communications*, 2004.

[18] H. Chen, A. Joshi, and T. W. Finin. Dynamic service discovery for mobile computing: Intelligent agents meet jini in the aether. *Cluster Computing*, 4(4):343–354, 2001.

[19] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz. An architecture for a secure service discovery service. In *Mobile Computing and Networking*, pages 24–35, 1999.

[20] C. Dabrowski, K. Mills, and J. Elder. Understanding consistency maintenance in service discovery architectures during communication failure. In *Proceedings of the third international workshop on Software and performance*, pages 168–178. ACM Press, 2002.

[21] E. Guttman. Service location protocol: Automatic discovery of ip network services. *IEEE Internet Computing*, 3(4):71–80, 1999.

[22] L. Iftode, C. Borcea, N. Ravi, P. Kang, and P. Zhou. Smart phone: An embedded system for universal interactions. In *Proceedings of the tenth International Workshop on Future Trends in Distributed Computing Systems*, 2004.

[23] T. Kindberg, J. Barton, J. Morgan, G. Becker, D. Caswell, P. Debaty, G. Gopal, M. Frid, V. Krishnan, H. Morris, J. Schettino, B. Serra, and M. Spasojevic. People, places, things: Web presence for the real world.

[24] M. Nidd. Service Discovery in DEAPspace. In *IEEE Personal Communications*, August 2001.

[25] P.Boddupalli, F.Al-Bin-Ali, N.Davies, A.Friday, O.Storz, and M.Wu. Payment support in ubiquitous computing environments. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications*, 2003.

[26] A. Ranganathan, R. E. McGrath, R. H. Campbell, and M. D. Mickunas. Ontologies in a Pervasive Computing Environment. In *Workshop on Ontologies and Distributed Systems*, 2003.

[27] N. Ravi, C. Borcea, P. Kang, and L. Iftode. Portable Smart Messages for Ubiquitous Java-enabled Devices. In *First International Conference on Mobile and Ubiquitous Computing*, pages 412–421, 2004.

[28] I. Satoh. Location-based services in ubiquitous computing environments. In *IEEE International Conference on Communications*, 2004.

[29] Z. Wang and J. Seitz. Mobile agents for discovering and accessing services in nomadic environments, 2002.

[30] M. Weiser. The computer for the twenty-first century. *Scientific American*, September 1991.

[31] F. Zhu, M. Mutka, and L. Ni. Splendor: A secure, private, and location-aware service discovery protocol supporting mobile services. In *First International Conference on Pervasive Computing and Communications*, 2003.