Creating Virtual Devices by
Accessing Sub-Components Across Wireless Devices
By: Hang Cheng
Advisor: Richard Yang

Abstract:

The advances in wireless technology in recent years almost guarantee its prevalence in the future. For example, Bluetooth, a short range wireless technology, can fit entirely on a chip that is smaller than a dime and cost less than one dollar to manufacture. Most new cellular phones already have or will soon have Bluetooth functionalities and other devices will soon follow. It is fairly obvious that future users will have enjoy greater mobility once they have been freed from the limitations imposed by wiring. However, wireless technology is also about being able to connect everything. Any two Bluetooth enabled devices should be able to communicate with each other over Bluetooth. If this universal connectivity is exploited to expose the devices’ other functionalities, one can theoretically connect the various sub-components together and create a new hybrid device.

The purpose of this project is to implement this idea and explore the implementation’s potentials and limitations. The implementation must be such that other programs utilizing the services provide by the implementation should not be distracted by its variegated nature. In other words, the two devices must come together fairly seamlessly.

This project’s prove of concept is for a laptop without an Internet connection to access the Internet by borrowing another laptop’s Internet connection via Bluetooth. While Bluetooth was chosen for this project, the implementation is largely independent of the wireless technology. The details of the underlying connection is hidden by the implementation. Any programs that want to utilize the connection should be able to do so with minimum overhead.

The vast majority of the implementation was coded in Microsoft C#. C# was chosen because of the large body of existing classes and libraries and its supposed interoperability across Microsoft operating systems. Given the presence and even dominance of Microsoft in most segments of computing, this should allow the implementation to easily migrate to other devices. The Bluetooth functions, however, are implemented in Visual C++ because the Microsoft Bluetooth drivers are only available in C++. Nonetheless, that part is only exposed via a network socket and all the Bluetooth details are hidden.

While the project did reach its goal of being able to browse the Internet via Bluetooth, certain limitations became very apparent in the process. These limitations, however, are largely the result of other programs running on top of the implementation.
Specifically, Internet Explorer is never designed for such an environment and its performance suffered as a consequence.

Introduction:

One of Sun Microsystems’s slogan was “The Network is the Computer”. Conversely, “The computer is a network” is equally true. One can think of the computer as a network of components that are each responsible for a certain task with the supporting hardware providing the connecting. In fact, terms such as “bandwidth” are also used in describing motherboards. Likewise, a 3-tier model with a database, a business logic layer, and a user-presentation layer communicating over a network appears to the end-user as a single entity. Thus, any device can be thought of as a network of components that can be modified and used for purposes other than what it was originally designed for. This idea has largely been limited in the past because of the need for physical connections between these components. For example, suppose that a computer has a very fast processor but limited physical storage while another computer has a slow processor but a much larger hard drive. While one can conceivably take the faster processor, put it in the computer with the large hard drive, and create a better computer, this is not a practical solution for a few reasons. First, the processor might not be compatible with the supporting hardware. Next, most users lack the skill to accomplish this and it is still a hassle in any case. Lastly, the end product of this operation is one better computer and one worse computer. Thus, the idea of borrowing and mixing various sub-components of devices to create a new device has largely been limited by the physical limitation of doing so.

However, if the underlying details can be hidden away so that only a well-defined, universal interface is exposed, the compatibility issue largely goes away. Furthermore, if the sub-components from different devices can communicate with each other without any physical alteration, the user will be spared the hassle of physically modifying the devices. Lastly, if new devices can be created from existing devices without destroying those devices, then it is no longer a zero-sum game of sacrificing something to create something better. These are the issues this project is trying to address by leveraging Bluetooth technology.

Bluetooth started out as an initiative to free the users from wiring. It is now an open-standard adopted by most of the members of the computing and other-related industries. At this time, Bluetooth is most often seem as part of cellular phones and computer peripherals such as wireless mouse and keyboards. While Bluetooth shares the same frequency range as IEEE 802.11 or WiFi, it is considerably shorter in range and lower in bandwidth. Most Bluetooth devices are generally limited to one megabit per second and has a range of about ten meters. This is usually adequate for the devices that Bluetooth is intended to connect. This limitation is one of questions probed by this project.
Bluetooth also includes a service discovery protocol and various profiles for connecting devices. These profiles are intended to guarantee the interoperability of devices implementing them. The SDP is intended to help these devices find other devices running the desired profiles or services. While the Bluetooth SDP is used in this project, it is really incidental and was used only because it is a necessary step to connect Bluetooth devices. Perhaps in the future, it will be better utilize to some advantage.

The universal nature of Bluetooth is the only significant aspect of the technology leveraged in this project. Essentially, since many devices currently have Bluetooth and even more devices will have it in the future, the sub-components of these devices can all be exposed via Bluetooth for other uses. The only limitation is how much access the operating system allows. A quick survey of the various operating systems suggests that the level of access generally offered is enough. For example, Microsoft Windows XP offers access to the file system, other processes, the memory, and communications stacks down to the IP level if desired. The various components can also be accessed by invoking the drivers for these devices. In fact, the Bluetooth connections in the project are invoked via the Microsoft Bluetooth drivers.

Related Research:

In *Rajicon: Remote PC GUI Operations via Constricted Mobile Interfaces*¹, the authors proposed and demonstrated a way for a 3G cellular phone to control a remote computer using a limited GUI. While their work is superficially similar to the proposed concept, their ultimate purpose is different. The authors created a set of programs that resides on the phone, a server, and the remote computer. The user issued commands over Japan’s 3G network to the server that operated the computer remotely via the Internet. The computer sent the results of executing those commands back to the server, and the server tailored the results before sending them back to the phone. The authors were concerned with the usability of the user interface and performance of the system. Any general concept for universal device interaction over a short range and composite functionality is not presented.

In *A Comparison Of Service Discovery Protocols And Implementation Of The Service Location Protocol*², the authors compared various service discovery protocols currently available. It is mentioned in the paper that Bluetooth contains a simple service discover protocol, thus making it a suitable choice for this project.

The idea of sub-components interacting and creating a new system is nothing new to the field. The advent of XML, web services, and SDPs were largely to deal with this issue. XML has allowed different computers of different businesses to communicate with each other and form a system for purposes such as ordering. However, these technologies are usually implemented at the Internet level between computers and servers. The purpose of this project is to allow the components of devices to do something similar.

Implementation:
The implementation can be divided into three modules and they communicate with each other via network sockets for modularity reasons:

The Relayer module is the heart of the implementation. The purpose of this module is to aggregate all the traffic coming from various services that want to communicate with its counterpart on the other device, deliver the data across to the Bluetooth module, and route the return traffic from the Bluetooth module to appropriate services. In other words, the Relayer is responsible for multiplexing and de-multiplexing of traffic to and from the Bluetooth module. This module is specifically designed to impose as little overhead as possible on the programs that want to use it. In order to route the traffic correctly, any program using the Relayer must have a twelve character unique identifier. This identifier, usually referred to in the source code as FID, is analogous to a port number. Any program that uses the Relayer must first connect to the Relayer on port 37337 and send the FID. The Relayer creates a Listener thread dedicated to the program. It also records the FID to socket mapping and saves it to the Directory. The Directory is basically a hash table that maps FID to sockets.

Anytime a program wants to communicate with its counterpart on the other device, it simply sends data through the socket it established earlier with the Relayer. The Listener thread for that program will receive the data and store it in a Data Unit that includes the FID of the program and the length of the data. The Data Unit is then added to the Data Handler. The Data Handler is a basically a queue that is shared by all the Listener threads and the Forward Out thread. All the traffic from the programs are aggregated into a single queue. The Forward Out thread repeatedly checks the Data Handler for available data and sends the available data as a packet to the Bluetooth module. The packet consists of a 12-byte FID followed by a 4-byte size field and a variable number of bytes for the data as specified by the size field.

Data from the Bluetooth module is received by the Forward In thread as a packet. The thread checks the FID of the packet and queries the Directory for the appropriate socket to send the data to. Information such as FID and length of the packet is stripped before sending it into the socket. Essentially, other than registering its FID with the Relayer at the beginning, the programs using Relayer can send and receive data as though it was an ordinary network socket between its counterpart on the other device. The only limitation is that services using the Relayer can only speak to its counterpart on the other device via the Relayer. Any cross communication between the services must be accomplished by a program that sits on top of these services. This is in line with the earlier constraint that these services must appear to programs using them as though the service and its counterpart are running on the same device. These connections can be thought of as virtual wires between the components on the devices.
The client and server modules are very loosely coupled with the Relayer module. The client is how the user interacts with the program and controls what services are running. The client program queries and requests from the server for services it can run, queries the other device for services it is running, queries and requests from the server for programs to utilize services running on the other device, and starts and stops services. For the purpose of this project, the client is also able to upload service and client programs to the server.

Each device has a file called “attributes.txt” that specifies its attributes. These attributes correspond to components the device can share or use to provide service. The client program reads this file and queries the server with these attributes. The server searches for existing programs that can utilize these components and returns the results. The client then requests the file and can then run it. Whenever the client runs a program, the process is recorded for future shutdown and checking purposes. Next, the client on the other device can query for services currently running. This is done via Bluetooth using the Relayer. The client program has an unique FID and must register with the Relayer when it starts. Once the client knows what services are running on the other device, it can query the server for clients to those services so it can utilize the service.

The Bluetooth module is the only part that is written in C++. The code is heavily borrowed from a site on the Internet. Its purpose is to simply take traffic from its network socket and send it down the Bluetooth connection and vice versa. The only part that is exposed to the other modules is its network socket.

The Bluetooth module are slightly different on the two devices. One of the device must run the server version of the module and the other must run the client version.
When the server version starts, it creates a Bluetooth query set that specifies the parameters of its service such as the UUID. This is then posted to the Bluetooth SDP and the server waits for an incoming connection.

The client version first queries for all Bluetooth devices within range and receives a list of them if there is more than one. Next, it also creates a query set and searches for the devices that has a matching query set which means having an identical UUID. For this project, the client takes the first matching device and requests a socket. From this point on, the socket is essentially the same as any network socket. The two version of the Bluetooth modules behaves exactly the same afterwards.

The three modules can be thought of as a stack. The Bluetooth module is the link layer, the Relayer is akin to the network layer, and the client module is just a program that runs on top of that. Consequently, they must also be started in this order because each layer relies on the layer underneath.

For this project there are also two pairs of programs created to demonstrate and explore the capabilities and limitations of the actual implementation. The first pair is the Forwarder and Forwarder-Client program. Their purpose is to give devices without an Internet connection or access to the server a way to access the server for querying and downloading programs. The Forwarder runs on the device with access while the Forwarder-Client runs on the device without access. The Forwarder-Client occupies the same port as the server, 7337, and essentially pretends it is the server. It accomplishes this by forwarding all incoming traffic to the Relayer which sends it to the other device. The Forwarder receives the traffic then forwards it to the server. The reverse happens when the server replies.

The other set of programs is the proxy server. It implements the SOCKS4 protocol. However, it was found that neither Internet Explorer nor Mozilla Firebird conforms to the specifications stated in the paper. The actual implementation of the SOCKS4 protocol was altered to fit the needs of the browser. The programs consist of Proxy and Pseudo-Proxy and they are similar to the Forwarder in functionality. Pseudo-Proxy appears to be a SOCKS4 proxy server to the browser. However, it simply sends everything it receives from the browser through the Relayer and vice versa. Proxy receives data from the Relayer and forwards the traffic to the appropriate Internet server. Proxy is also searching for the SOCKS4 “connect” packets that state which server it needs to open a connection to. Upon receiving the reply from servers, it sends the data through the Relayer which sends it back to the Pseudo-Proxy. It is very important to note that this implementation of a proxy server is only single threaded. The limitation here is that the Relayer only allows one FID per program. Thus, to keep things simple, the proxy program was limited to only one thread and one connection with the browser at only one time. This becomes important later on.
Results:

The Bluetooth and Relayer modules worked flawlessly. The Bluetooth module requires a few seconds to finish SDP and to connect. The Relayer was always able to multiplex and de-multiplex the traffic correctly and in a timely fashion. These two modules caused no trouble nor any significant latency. Thus, the most essential parts of this project worked very well. However, it must also be noted that the test was a prove of concept and not a test of reliability. Therefore, these two modules were never seriously stressed.

The client and server modules worked well for most of the time. The server module is very robust and is able to handle multiple connections and exceptions. The client, however, was not robust and occasionally froze during the transfer of files. The probable reason for this is that both server and client were implemented using C# network streams that may not be suitable for the purpose of sending raw data. The Relayer, in contrast, was implemented using network sockets, one layer below the streams, and did not experience any problems. Furthermore, the Forwarders occasionally had problems forwarding traffic from a client to the server and back over Bluetooth probably for the same reason.

The project succeed in reaching the goal of being able to access the Internet via Bluetooth by borrowing an device’s Internet connection. Internet Explorer was set to use a SOCKS4 proxy server at “127.0.0.1” on port 37380. While Internet Explorer was able to load most of the requested pages correctly, there were some notable limitations. First, the connection and render time varied from a few seconds as in the case of Google to a few minutes or more for other sites. The notable difference between Google and other sites is the lack of advertisement and the centralized location of the necessary images. As a result, the pages returned by Google only required a few connects and disconnects. Once the proxy has connected with a server, the transfer is relatively fast. Reconnection
and connection to a new server is responsible for most of the latency. As noted earlier, the proxy program is single threaded. Therefore, Internet Explorer was unable to open multiple connections. As a result, it had to wait for each connection to time out before requesting a new connection. This limitation was not a problem for sites like Google but sites like CBS.com had many different sources for each page that it required a few minutes to request connections, download, and wait for time-outs before requesting another connection. Essentially, single source downloads was not a problem at all. In fact, once CBS.com was loaded, Internet Explorer was able to view the streaming videos without any problems. The lost in download speed was very small.

Another limitation encountered was the lack of a substitute for access to a DNS server. Therefore, the IP addresses of sites visited had to be written into the “hosts” file prior to the visits. The network adapter was left enable for DNS resolution during the streaming video test. However, since all the packets were transmitted over Bluetooth, this had no influence on the performance.

![Google](http://www.google.com)

Fig. 5: [http://www.google.com](http://www.google.com) loaded over Bluetooth
Discussion:

The results suggest that the limited bandwidth of Bluetooth is not really an issue. Bandwidth is not a crucial factor in controlling components over a wireless connection. Most of the control packets such as those to initiate an outside connection or to retrieve something from a server are relatively small. The only time the buffers were filled to the preset maximum of five kilobytes was during the transfer of images and other large chunks of data. Response time, on the other hand, should be an important factor because the goal of this project is for it to appear as though all the components were on the same device. In weighting these two factors, Bluetooth is adequate for most things with the possible exception of file transfer and storage.

Most of the difficulties came from the programs running on top of the Bluetooth module and the Relayer module. The Relayer presented programs using it the constraint of having only one socket. As a result, Internet Explorer was unable to open up multiple connections and fetch items off the Internet simultaneously. It had to wait and time-out instead before a new connection is made and new items are fetched. This problem, however, can be remedied in a few ways. First, the proxy programs can be written so that pseudo-proxy would allow Internet Explorer to open multiple connections. Once it has received the traffic, it can aggregate all the packets and send them over to proxy as a single stream. The proxy server would then open as many connections as Internet Explorer requests. This means Internet Explorer would issue new connect commands as
it needs without being limited to a single connection at time and waiting for time-outs. However, this requires pseudo-proxy to keep track of which packet coming back from proxy is intended for which socket of Internet Explorer. In other words, it has to multiplex and de-multiplex. Another solution would be to change Internet Explorer’s time-out to a lower value so that it would time out faster. Alternatively, a new browser that is designed for single threaded connections can also be created specifically for this kind of environment. Lastly, web sites can be optimized so that all the items can be loaded without a new connection to another server. It should be noted that all these problems are not the result of Bluetooth or Relayer but rather the result of programs and websites ill-suited for this environment.

The project was largely a success as a prove of concept. The laptop with only a Bluetooth connection was able to connect to the other laptop, query the server via the other laptop, download the Pseudo-Proxy program, run it, and then access Google using Internet Explorer. The lack of DNS resolution was a notable limitation. The actual performance and reliability of the implementation was not tested. While Internet Explorer was able to play streaming video without any problem, it is unclear how well and reliably this system will work once multiple programs run on top of the Relayer.

Future Work:

One area that was never tested is the possibility of combining more than one device. The Bluetooth modules are capable of accepting more than one Bluetooth connection. In fact, Bluetooth was designed to form small personal networks of devices. The Relayer module can be easily modified so that multiple instances of it can run and allow the routing of traffic to multiple different devices.

The Bluetooth and Relayer modules should also be stress tested and optimized for better reliability and performance. Once many components are shared the limited bandwidth of Bluetooth can become a bottleneck. Thus, optimization should be done to better utilize it.

To fully reach the goal of being able to browse the Internet via Bluetooth, either a better suited browser must be written or the proxy programs must be rewritten to support multiple connections. Furthermore, DNS over Bluetooth must also be implemented.

Lastly, the Relayer and Bluetooth can also be rewritten so that it does not continuously poll for new data and waste CPU cycles. Early versions of the proxy program was limited by this problem because various modules would content for CPU cycle along with the browser. This resulted in many images timing out. This problem was remedied by forcing threads to go to sleep for a few milliseconds. However, an event driven model can avoid this problem entirely and make the interaction between the modules much smoother and more reliable.
1 Su N. M., Sakane Y., Tsukamoto M., Nishio S., Rajicon: Remote PC GUI Operations via Constricted Mobile Interfaces, Mobicom 2002

2 Bettstetter C., Renner C., A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol, In Proceedings EUNICE 2000, Sixth EUNICE Open European Summer School, Twente, Netherlands, 2000
