BLUE NOMAD
AN INVESTIGATION INTO FILESHARING WITH MOBILE DEVICES

By

MATTHEW CARROLL

A PROJECT IN LIEU OF THESIS PRESENTED TO THE COLLEGE OF FINE ARTS

OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF ARTS

UNIVERSITY OF FLORIDA

2012

To Congressman Ron Paul for fighting the power that would silence innovation, communication, and true education.

ACKNOWLEDGMENTS

I thank my mom for helping to bankroll an extra two years of higher education, hopefully the ROI of this publication pays dividends. I also thank the many bloggers and forum junkies of the world who helped make this effort a reality.

TABLE OF CONTENTS

page

LIST OF TABLES

Table page

LIST OF FIGURES

Abstract of Project in Lieu of Thesis Presented to the College
of Fine Arts of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Arts

BLUE NOMAD
AN INVESTIGATION INTO FILESHARING WITH MOBILE DEVICES

By

Matthew Carroll

May 2012

Chair: Benjamin DeVane
Major: Digital Arts and Sciences

Many experts agree that the future of computing is mobile, in fact, many expect to see computing become pervasive. Today, a growing number of people engage in mobile computing, but this process is largely solitary and represents a temporary extension of their primary desktop computers. Movement towards true mobile computing and especially pervasive computing requires a social shift in how and where people use computers. Blue Nomad is an effort to jumpstart that social shift.

Blue Nomad constitutes a proximity based information exchange system that offers file-sharing over Bluetooth connections. The system's immediate purpose is to easily move files from a mobile device to a nearby desktop or laptop computer without physically connecting anything. While other Bluetooth file transfer systems often require that two devices "pair" with each other by exchanging a secret pass-code, Blue Nomad does not require this pairing – no setup is required. Not only does this ad hoc nature make it easier to move your own files around, it opens the door to anonymous sharing and social integration of users. The project features three implementations of Blue Nomad: an Android implementation, a Java SE/BlueCove implementation and a Python/PyBlueZ implementation. The Android implementation represents a "Nomad" in

the system and the Java/Python implementations represent “Oases” in the system. The primary difference is that Nomads are mobile whereas Oases are stationary. With a single Nomad implementation combined with a single Oasis implementation, the core behavior and potential of proximity based information exchange are visible in the Blue Nomad project.

CHAPTER 1
VISION

Nomadic Computing

Many experts agree that the future of computing is mobile, in fact, many expect to see computing become pervasive. Today, a growing number of people engage in mobile computing, but this process is largely solitary and represents but a temporary wireless extension of their primary desktop and laptop computers. Movement towards true mobile computing and especially pervasive computing requires a social shift in how and where people use computers. This project is an effort to jumpstart that social shift.

In this paper I describe Nomadic Computing as a form of social computing in which users travel with significant digital media, share it with others on the road, and broadcast it to listeners. Going further, I introduce a location-based media-sharing system, which I developed for digital devices that use the Bluetooth wireless communications standard. The term for this new computing paradigm is derived from the lifestyle of nomads traveling along the Silk Road:

> ...trade would take place in discrete bundles of goods being carried over short distances, from one city to the next (Rowan, 2006).

Nomadic Computing is about proximity. Users do not download files from isolated, remote servers scattered around the world. Instead, users download files and stream media from other users in the vicinity. In this manner, Nomadic Computing fundamentally integrates social interaction. It is hypothesized that local interaction of mobile devices will result in an increase in mobile computing, and greatly expand the variety of activities pursued on mobile devices.

Average computer users understand very little about network layouts and protocols. It is unlikely that technical details alone would incite a shift in social perception of mobile computing. This project introduces Nomadic Computing as a thematic wrapper for this new location-based paradigm of social computing. In keeping with this theme, additional artifacts within this system extend the nomad analogy.

Nomads

Mobile devices participating in Nomadic Computing are termed Nomads. Historically, nomadic people have been defined as having no permanent home - much like mobile devices which travel from a user's home, to work, to recreational areas, and then back again. There is no location where a mobile device is more "at home" or "appropriate" than another. Therefore, mobile devices are quite nomadic.

The nomad analogy in this paradigm specifically targets nomadic traders along the Silk Road. These nomads would travel independently while moving goods. On the road nomads would meet, band together, and then disperse - sometimes they would share goods and supplies along the way. Like the Silk Road nomads, a Nomad in the Nomadic Computing paradigm also carries goods and supplies, it too travels along its own path while temporarily interacting with other Nomads, and it too may send and receive goods and supplies. Hence, a mobile device in the Nomadic Computing paradigm is exactly what a nomadic trader was on the Silk Road.

Oases

Along their trading routes, a nomad would stop occasionally at small sanctuary towns known as oases. Oases, too, have an analogy in Nomadic Computing. A

Nomadic Computing Oasis is a stationary server which offers goods and supplies at an established location. In this manner the act of exchanging goods can take place even in a drought of other Nomads. An Oasis could be used for many different purposes - it could allow Nomads to both submit and receive goods (like a trading post), to receive specialized goods offered only by that Oasis, generate goods on the fly, etc.

Though an Oasis would probably be implemented using a standard desktop or laptop computer, it would still continues the Nomadic Computing paradigm. Regardless of platform, an Oasis is predicated on proximity and must offer services without the presence of a World Wide Web. Hence, an Oasis is better described by its services than its physical state.

Mobile device Nomads combined with stationary Oasis servers complete the foundation of a landscape which promotes Nomadic Computing. With these structures in place, mobile computing can begin to present itself outright as an activity independent of desktops and laptops. This social shift in computing moves society closer to pervasive computing, the logical conclusion of ever-smaller and ever-cheaper electronics.

Social Implications

Nomadic Computing together with an increase in mobile proliferation has great potential to make a significant social impact. With widespread localized sharing of goods, one must consider the activities which might result. In a world driven by Nomadic Computing, the way data moves through society would change, the regulatory structures would change, and the proliferation of information would change.

To understand the potential impact of Nomadic Computing, one must understand the short-comings of the current global network model of the Internet. Currently, the internet allows us to beam content from any one location to any other location directly. This transport infrastructure is incredibly quick and efficient; it has revolutionized the way information travels and how people, businesses, and governments operate. However, the Internet can have two potentially significant negative results:

- The individual becomes obscured, and non-trivial social interaction tends to decrease with use of the internet.
- Despite the goals of the developers of the internet, the network which supports the internet as we know it is easily monitored and controlled by governments and other interested parties. The government knows what you are accessing and what you are downloading. The government also has the ability and willingness to block your access to the internet.

In healthy social contexts, Nomadic Computing may increase interpersonal social interaction. Users may begin messaging, or chatting with, or meeting other users with similar interests (prompted by the music, movies, and books that a user is sharing). Today there are an unprecedented number of relationships which begin via social media - Nomadic Computing personalizes this experience even further. Many people complain about the de-humanizing nature of constant computer interaction - Nomadic Computing may begin to reverse this trend.

In an unhealthy social context, such as one involving an authoritarian government, Nomadic Computing provides a fundamental silver bullet to information control. Currently the Internet is hailed as the medium for unstoppable information

proliferation - but this is not really the case. The Internet is the best option society has, but society can do better.

Many governments already control, or are working to control the Internet. As an example, consider the Stop Online Piracy Act (SOPA) and Protect IP Act (PIPA) bills which were addressed in the U.S. House and Senate respectively ("Sopa and pipa," 2012). Had those bills passed, the U.S. government would have legally obtained unilateral authoritative control over the internet (the U.S. Congress continues to review altered versions of SOPA and PIPA) ("Even worse than," 2012). That is how easy it is to control the Internet. It is not technically cumbersome to take over the Internet, it is just that pesky ideas like individual liberty are currently standing in the way. Once those hurdles are passed, there will be no preventing tyrannical governments from obtaining total control over the Internet ("Protect ip /,"). Enter Nomadic Computing. Nomadic Computing takes place within a local vicinity and does not require any special equipment to operate. A central government or agency could never track the activities happening around you because there is no global access point to what you are doing (as opposed to the Domain Name Servers required for the Internet). Therefore, Nomadic Computing really is a silver bullet to the transgressions of an authoritarian government against information exchange.

The future of information exchange in the US might be in danger, but some governments already censor and control the Internet. One can simply look at the present-day Middle East. In a handful of countries in the Middle East the Internet is already controlled and/or blocked. Social media is blocked specifically for the purpose of

deterring uprisings. With Nomadic Computing, it would be impossible for these governments to block information exchange. As long as sources of information could get within a certain proximity of those needing information, the exchange would be inherently unstoppable. Therefore, Nomadic Computing could play a major role in democratization of nations all over the world.

It is unclear how quickly or fully the Nomadic Computing paradigm might be embraced. However, if it were embraced fully, it could have a considerable, lasting impact on societies all over the globe.

CHAPTER 2
PROJECT SCOPE

Affecting the global social paradigm of computing requires about 7.5 months, give or take. Unfortunately, this project only afforded about 6 months of work, therefore it falls short of the general goal. This project focuses exclusively on an Android-based file sharing system over Bluetooth communication. The application produced is aptly named Blue Nomad. The goals of this project explicitly include:

- Runnable application on Android devices (API 2.2 and above)
- Ability to connect to other devices running Blue Nomad in the vicinity
- Ability to query a name and personalized thumbnail from connected devices
- Ability to query available files on connected devices
- Ability to download any offered file from a connected device

CHAPTER 3
COMPARATIVE PROJECT ANALYSIS

Nomadic Computing brings together a number of hot topics in the world of software. First, Nomadic Computing is heavily based on the act of file sharing, a subject commanding a notorious history over the last decade. Second, Nomadic Computing either generates or utilizes ad hoc networks. Third, Nomadic Computing integrates a social aspect with the possibility of becoming a pervasive system.

Napster

Napster jumpstarted a file sharing revolution in the year 2000. Napster provided a free website and application client which connected users all over the world and facilitated the sharing of music files (Tyson, 2000). This filesharing system utilized the Internet as a network backbone for file transport; it utilized individual users' machines to host files.

The Blue Nomad project is also based on the action of file sharing, but involves two major differences. Blue Nomad allows for sharing of any type of file, and Blue Nomad utilizes Bluetooth networking for connectivity.

BitTorrent

BitTorrent became popular a couple of years after the collapse of Napster. BitTorrent is actually a protocol which allows files to be downloaded in chunks and then reassembled on the client's machine. Unlike Napster, BitTorrent is applicable to any type of file, but it still uses the Internet as a connection layer ("A history of," 2011).

BlueTorrent

BlueTorrent is a project that was developed at the University of California, Los Angeles in 2007. The description of the project's goals are very similar to Blue Nomad:

> Using BlueTorrent, people can share audio/video contents as they move about shopping malls, airports, subway stations etc.
>
> (Jung, 2007)

Despite the mention of social interactions, the BlueTorrent project focuses largely on low-level Bluetooth discovery processes which were required for easy networking. Additionally, BlueTorrent did not deploy to end-user devices such as Android phones. Both of these areas are addressed by the Blue Nomad project.

Facebook

Facebook is a social networking platform founded in 2004 by Mark Zuckerberg while at Harvard University. The platform, similar to systems like MySpace, provides a platform to connect with friends online. Through the evolution of the Facebook platform, the social network now collects, organizes, and presents information about likes/dislikes/event attendence/brand opinions, etc. for the purpose of enriching social connections.

The Nomadic Computing vision looks to offer a spin on this social network. Nomadic Computing would quite literally create a social network of devices. But, unlike Facebook, what users like and dislike would not be entered by the user, but synthesized from records of media downloaded and choices made. From the chaotic interactions of massive ad hoc networks, there would arise individual and social patterns of behavior which could then be used to connect compatible users for various purposes. Therefore, where Facebook builds around existing relationships, Nomadic Computing provides a means for discovering new relationships.

CHAPTER 3
DEVELOPMENT

Development of Blue Nomad took place in an evolutionary manner. This was my first Bluetooth-based project, therefore I did not have the experience necessary to design the application components before developing them. Instead, individual features were designed and implemented one by one, integrated into the project piecemeal. Occassionally, code was refactored to correct obvious inefficiencies and other issues. Due to the evolutionary process used, development is first described in chronological steps and then described in terms of design.

Development Evolution

1st. Development started on a Droid X2 phone running Android 2.3. The purpose of this starting point was to understand the process of discovering and connecting to other devices. At this time an ASUS Transformer tablet was also available for testing which provided a device to connect to.

Exploration of device connectivity began with exploration of the available Android Bluetooth API. The Java objects provided for Bluetooth connectivity were straightforward, but their Android interactions were a bit unusual. Namely, when certain Bluetooth behaviors were requested, the results were not provided in a return value, or a function callback. Instead, special Android Activity behaviors were invoked to handle the results. It took a little while to understand the communication paradigm for Bluetooth on Android, but it now appears quite simple. The general steps for discovering/connecting to devices were:

1. Start Device Discovery
2. Collect All Nearby Devices
3. Cancel Device Discovery

4. Run Service Discovery On All Devices
5. Discard Devices Without Blue Nomad Service
6. Open Sockets To Remaining Devices
7. Use Blue Nomad Service

With the above steps working, I found myself with an open socket and no idea about what to send through it. The Android coding was tabled and an exploration of protocols began.

2nd. Networking protocols were something I had never worked with before. I had never taken a class in network programming, nor had I worked with sockets previously. I wasn't sure if I should send simple text requests between devices, use a somewhat more sophisticated syntax like JSON or XML, or if I should wrap the sockets with an entire 3rd party library to handle communication. I began by considering JSON and XML.

XML was ruled out quickly as a communication protocol because of its complexity. I had worked with multiple XML parsers before - some are simple, like ActionScript's E4X parser (The e4x approach), others seem unnecessarily complex like the Xerces C++ parser ("Programming guide,"). Knowing that I would implement this behavior across multiple platforms, I did not want to risk pouring time into bloated XML libraries.

JSON was the next protocol I considered. I was very close to using JSON because I had previous experience with JSON in the context of client-side web design. The JSON protocol is very simple, easy to parse, and easy to understand. The question that closed the door on JSON was about how I needed to handle binary data. I didn't know how I was supposed to transport large amounts of binary using JSON. To illustrate the problem, consider the following possible JSON object describing a file for a Blue Nomad transfer:

```
{
        'filename':'my-HD-Image.jpg',
        'size':582473450,
        'data':[ASSUME BINARY HERE]
}
```
Figure 3-1: Example JSON Object

Notice the closing curly-brace at the end of that object. To parse the above JSON all at once, all of the text through the closing curly-brace would need to be in memory at one time. Obviously, if a file of any considerable size were transferred, the sum of its binary data would not fit in memory all at once. The only solution then would be to parse and interpret the JSON as a stream as it transfers into the device. This was probably possible, but for the same reason as the XML, I decided to look for a simpler solution.

When the parsing complications of XML and JSON failed to pan out, I looked for a higher level solution. The first stop was OBEX - OBject EXchange protocol for Bluetooth http://www.wisegeek.com/what-is-obex.htm. OBEX is used for Bluetooth file transfer in existing systems, it seemed like a direct solution for Blue Nomad. Unfortunately, searching for an Android OBEX library that I could plug into a socket did not generate any results. I could have implemented OBEX, but the protocol appeared to be more complex and troublesome than either XML or JSON. Therefore, I moved past OBEX.

After OBEX I stumbled upon a special Google tool: Protocol Buffers ("Developer guide,"). Google explains their product best:

> Protocol buffers are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data – think XML, but smaller, faster, and simpler. You define how you want your data to be structured once, then you can use special generated source code to easily write and read your structured data to and from a variety of data streams and using a variety of languages – Java, C++, or Python ("Protocol buffers,").

Protocol Buffers sounded great - exactly the kind of tool I needed. I could configure my data once and then port it to multiple languages, and the information was

available in "a variety of data streams." I was all but ready to dive head-first into protocol buffers when I decided to just take a quick look at the existing File Transfer Protocol(FTP).

Early on in my FTP research I found a well written beginner's article on FTP (Koneri, 2004). Quickly, I realized just how simple the FTP protocol was. The FTP protocol is delimited by new-lines which make it trivial to separate one command/request from another. The format of each server response is the same and begins with a status code to immediately communicate success/failure of a request. Binary is transferred by opening a second port and just shoving the data through without special markup or packaging. As useful as protocol buffers appeared, creating my own FTP analogy just seemed much faster; so that is what I did.

3rd. By the time I chose to emulate FTP with my own protocol, I no longer had posession of the ASUS Transformer tablet. That meant I had no second Android device with which to connect and test. As a result, I decided to implement Blue Nomad between a desktop and a laptop before working on the Android version. Researching application-level Bluetooth libraries showed that the option with the shallowest learning curve was a Python library called PyBlueZ. I promptly wiped a desktop and laptop, installed Ubuntu 10, and began coding in Python against PyBlueZ (I used Ubuntu because the BlueZ Bluetooth stack ships in the core distribution).

PyBlueZ's simplicity made Android's Bluetooth API look like spaghetti, just completely disorganized. Within a few days I had the basics of the Blue Nomad protocol working between the desktop and laptop. It was not long before I had a Blue Nomad

client/server pair working well without any errors or issues. It was time to move to the next system.

4th. After completing client/server pairs in Python, it became clear that working with standard computers was much quicker and easier than developing for a mobile device. Clearly an Android implementation for Blue Nomad was required, but there was a way to make that job easier. Android is generally coded in Java, meaning that if I could find a simple Java Bluetooth library, the guts of the Android Blue Nomad system could actually be developed on computers and then adapted for the Android platform.

BlueCove was discovered when looking for a Java Bluetooth library. Unfortunately, the support channels for BlueCove appeared limited, but so were the library choices - I started exploring BlueCove. The BlueCove library immediately set itself apart from PyBlueZ by crashing before a single command was even issued. The library reported an error that the Bluetooth system failed to start. A bit of searching revealed that a special BlueCove snapshot build was required to work on a 64 bit Windows 7 machine (this time I was working in Windows). With the snapshot build downloaded, the library was able to start the Blutooth services and I was able to begin coding against it.

Continuing the PyBlueZ/BlueCove dichotomy, another mysterious issue presented itself when attempting to discover other devices. For some reason when device-discovery was started, it would immediately return and then error out. Again, I searched for solutions. Thankfully, this issue was even easier to solve than the previous one. It turned out that unlike PyBlueZ, BlueCove handles discovery asynchronously. That means the code must explicitly ask to wait around for the device discovery to finish, otherwise the code just continues processing and doing other things. Therefore,

sleep(1000) was looped until BlueCove was done discovering, at which point a number of devices had been collected.

With devices collected, it was time to find those offering the Blue Nomad service. Initially I hoped to be able to run a general service discovery across all devices, but BlueCove requires an explicit search of each device. Arriving at the third BlueCove issue, BlueCove simply refused to find any services in any of the discovered devices. I spent a couple of days trying to troubleshoot this issue, but I think I deserved it because I eventually found out that the cause was the same as the device discovery issue. Not only does BlueCove discover devices asynchronously, but it also discovers services asynchronously. The issue was frustrating, but I should have caught it quicker. At this point the application held a collection of devices offering the Blue Nomad service.

Finally, it was time to implement the Blue Nomad service in Java. The implementation was trivial as it was largely a port of the same logic already coded in Python. The only significant difference was moving from the Python file system API to the Java file system API, but both were easy to work with. I validated the Java implementation by connecting it to my Python client/server implementations on my other computers. A few bugs appeared, but they were squashed and the BlueCove/Windows 7 laptop was working flawlessly with two PyBlueZ/Ubunu machines. The Blue Nomad system was coming together nicely.

5th. Back to Android. A Java implementation of the Blue Nomad protocol was developed and tested, it was time to integrate it with the previous Android work. To double check that the Android system was still working like before, I tried to simply open a socket connection between my Droid X2 and my Windows laptop. It failed. A few

months earlier that connection was working just fine between the Droid X2 and the ASUS Transformer. Unfortunately, I no longer had possession of the ASUS Transformer to try and diagnose the issue. The Droid X2 saw the laptop, but failed to connect to the Blue Nomad service with a complaint of "Connection Refused."

Yet again I hit the forums. The problem was very worrysome because many posts were centered around a similar issue, but few questions were successfully answered, and none of the existing answers solved my problem. Without the Transformer, I could not be sure whether the problem was universal between any two devices, or if the problem was that Android devices simply could not connect to computers (this would have prevented completion of the project). Two frustrating weeks passed without a solution.

I noticed that my searches continued to link me to a particular forum post where the author illustrated a similar issue. He said his solution was to manually unpair other devices from his phone (Bluetooth devices can be paired for quicker connectivity). Over the two weeks, I paid no mind to his solution because I had never paired my phone with any other devices. However, I was out of ideas and decided to see if my phone was paired with anything after all. Amazingly, my phone was paired with all three of my machines: my Windows laptop and my Ubuntu desktop and laptop. I still have no explanation for this. I immediately unpaired all of those devices and 45 seconds later I had a successful socket connection between my Droid X2 and my Windows laptop.

Relieved and running short on time, I quickly implemented the Blue Nomad service for Android. A few changes to the initial BlueCove solution were required, but generally speaking the previous work was JAR'ed and included in the Android application without

issue. The file system was again the primary area of difference in implementing the Blue Nomad service. On Android all transferred files were placed in the official "Downloads" directory, and Blue Nomad icon images were placed in the official cache directory for the application. With Android file transfers working, the only remaining task was to polish the presentation a bit and clean up the code.

CHAPTER 4
DESIGN

The development chronology of Blue Nomad has been introduced. Now an overview of design decisions is offered for a number of the major development areas.

Blue Nomad Protocol

To exchange files and information between devices and machines, a communication protocol was required. The Blue Nomad protocol is a simple text-based protocol based very loosely on the characterization of the File Transfer Protocol (FTP). Like FTP, the Blue Nomad Protocol (BNP) sends messages from a server which begin with a status code followed by human readable information. BNP clients send messages which start with a request code, followed by pertinent data associated with the given request. BNP messages are delimited by the newline character, '\n'. In general a BNP client request looks like:

CLIENT_CMD arg1 arg2...\n
In general a BNP server response looks like:

XXX Human readable explanation.\n

When requested, binary data is also sent through the same channel as control commands. The receiver of the binary data is informed first with a control command that it will be receiving some binary data and the size of the binary data in bytes. Then the binary data is sent without any control code and without a delimiting '\n' character. The binary is simply expected to end after the given number of bytes have been transferred.

The following is an example protocol interaction. For illustration purposes the newline character is shown, and binary code is abbreviated as [BINARY]:

[client > server]:GET_ID\n
[server > client]:100 001060AA36F8\n
[client > server]:GET_NAME\n
[server > client]:100 My Server Name\n

[client > server]:GET_ICON\n
[server > client]:100 8643 jpg\n
[server > client]:121 Ready for file transfer.\n
[server > client]:[BINARY]
[server > client]:200 Data transfer complete.\n
[client > server]:GET_FILELIST\n
[server > client]:120 Starting file list.\n
[server > client]:221 15348 myImage.jpg\n
[server > client]:221 2456842 mySong.mp3\n
[server > client]:221 68137 myBook.pdf\n
[server > client]:200\n
[client > server]:CLOSE\n

Figure 4-1: Example Blue Nomad Protocol Communication

In plain english that interaction looks like:

[client > server]:Give me your unique ID
[server > client]:My unique ID is "001060AA36F8"
[client > server]:Give me your readable name
[server > client]:My readable name is "My Server Name"
[client > server]:Give me your user thumbnail image
[server > client]:My user thumbnail image is 8643 bytes in size with extension "jpg"
[server > client]:[IMAGE BINARY DATA]
[client > server]:Give me a list of available files
[server > client]:Starting list of files
[server > client]:myImage.jpg is available at a size of 15348 bytes
[server > client]:mySong.mp3 is available at a size of 2456842 bytes
[server > client]:myBook.pdf is available at a size of 68137 bytes
[server > client]:Done sending list of files
[client > server]:I'm leaving, CYA!!

Figure 4-2: Example Blue Nomad Protocol Communication In English

A full specification of the Blue Nomad Protocol is available in the Appendix.

Parsing

One benefit of such a simple protocol is ease of parsing. A client protocol interpreter need but check the first three characters of a given message to process the status code. All text after the status code can be stored for display. An interpreter can be implemented with as little as three public behaviors:

```
/*
 * Sends the content of "cmd" through socket
 * to server for processing
 */
BNPResponse sendRequest( String cmd )

/*
 * Waits for a reply from the server, then formats the
 * reply as a BNPResponse object and returns it.
 */
BNPResponse getReply()

/*
 * Reads "size" number of bytes of data from the server and
 * writes them to file using "writer".
 */
void getBinary( Writer writer, int size )
```

Figure 4-3: Blue Nomad Core Client Protocol Interpreter

In this API, BNPResponse is an object encapsulating a status code and an explanation. The Writer object represents a platform specific object for writing data (like to a file). An entire BNP client can be written with these three interpreter behaviors alone. The server logic is more complex but it too can use a similar protocol interpreter to handle communication responsibilities.

Python API

The Python Blue Nomad implementation is very basic because there was never going to be a need to extend the Python behavior. The only requirement was a functional client/server pair running atop PyBlueZ.

All Blue Nomad client/server logic is contained in a Python module called bluenomad.py. Outside of the bluenomad module, there are only 2 additional files: one file is used to start an instance of the Blue Nomad server, the other file is used to start an interactive instance of the Blue Nomad client. When running, each application's console appears as shown below:

Python Blue Nomad Server Output

```
Accepting clients...
listening on port 0
Accepted connection from ('00:02:72:BF:3A:B9', 1)
Created server thread
Started server thread
About to begin discussion with client
[interaction with a client]
Done talking to this client, closing socket.
Accepting clients...
listening on port 0
```

Figure 4-4: Example Blue Nomad Server Output

Python Blue Nomad Client Output

```
BLUE NOMAD @ 00:21:4F:FC:7F:62:10
connecting on 00:21:4F:FC:7F:62
What do you wanna do?
 1 - Get Nomad ID
 2 - Get Nomad Name
 3 - Get Nomad Image
 4 - Get Nomad Filelist
 5 - Get Nomad file
 9 - EXIT
[series of choices]
>9
PEACE
```

Figure 4-5: Example Blue Nomad Client Output

Java API

More thought went into the Blue Nomad Java API than the Python API. The goal of the Java system was both to have a functional client/server pair running on a Windows laptop, but also to be able to re-use as much code as possible in the Android app. The following is a breakdown of Blue Nomad Java classes by package:

com.s4d.bluenomad.protocol

The protocol package contains logic which depends only on the format and design of the Blue Nomad protocol. That is, the protocol package does not make assumptions

about whether Bluetooth or some other connection is being used, nor does the package make assumptions about whether streams or some other system is being used for data transfer.

BNProtocol.java

Defines constants for elements of the Blue Nomad protocol, like all possible commands and return codes.

BNResponse.java

Encapsulates a single Blue Nomad server response (includes a status code and human readable description).

Client.java

Defines API to request all of the things a Blue Nomad client can provide (ID, name, filelist, etc). Client is an abstract class which requires a subclass to choose the connection technology (IE Bluetooth). Additionally, the subclass must implement file system access for the desired platform. Additionally, Client requires a ClientPI to operate.

ClientPI.java

Blue Nomad protocol interpreter for client -> server communication. ClientPI is an interface which requires implementation for use. The specific implementation of ClientPI used determines the conduit for data transfer (IE streams).

Server.java

Provides an interface for starting and stopping a Blue Nomad server as well as getting/setting the server's ID and name. Server is an abstract class. Its implementation chooses what connection technology is used (IE Bluetooth).

ServerPI.java

Blue Nomad protocol interpreter for server -> client communication. ServerPI is an interface. Its implementation chooses the conduit for data transfer (IE streams).

com.s4d.bluenomad.protocol.bluetooth

Contains Blue Nomad classes pertaining exclusively to Bluetooth behavior.

BNBluetoothConfig.java

Defines constants for elements of the Blue Nomad Bluetooth implementation, like the Blue Nomad service name, and the Blue Nomad service UUID.

com.s4d.bluenomad.protocol.bluetooth.bluecove

Subset of Blue Nomad Bluetooth classes pertaining exclusively to the BlueCove architecture.

BluetoothDiscoverer.java

Helper class which makes BlueCove device and service discoveries synchronous.

Server.java

Implementation of a Blue Nomad server for the BlueCove platform.

com.s4d.bluenomad.protocol.desktop

Contains Blue Nomad classes pertaining exclusively to non-mobile platforms.

Client.java

Implementation of a Blue Nomad client which implements file IO using the standard Java API and also designates directories nomad-config and nomad-files to hold icon images and downloaded files respectively.

com.s4d.bluenomad.protocol.interpreters

Contains Blue Nomad protocol interpreters for various communication conduits (like streams).

StreamClientPI.java

Blue Nomad client protocol interpreter which communicates via streams (InputStreams and OutputStreams).

StreamServerPI.java

Blue Nomad server protocol interpreter which communicates via streams (InputStreams and OutputStreams).

com.s4d.bluenomad.tests.connectivity

Includes runnable classes for testing BlueCove functionality as well as running BlueCove clients/servers.

- BlueCoveBlockingDiscoveryTest.java
- BlueCoveConnectivityTest.java
- BlueCoveServerTest.java
- BlueCoveServiceDiscoveryTest.java
- ConsoleClient.java

Android API

The purpose of designing an intelligent Java solution for Blue Nomad was to minimize Android-specific workload. To that end, three packages from the generic Java system were JAR'ed and included in the Android project:

- com.s4d.bluenomad.protocol
- com.s4d.bluenomad.protocol.bluetooth
- com.s4d.bluenomad.interpreters

With the above code included, only two additional classes were required to implement the Blue Nomad protocol for android:

- com.s4d.bluenomad.protocol.android.Client
- com.s4d.bluenomad.protocol.android.Server

However, the Blue Nomad system was hidden by an overarching Service which provided a single access point to the Blue Nomad functionality. This was done primarily

to allow the Blue Nomad connections to persist even when a visual application was not running. However, using a Service provided the additional benefit of keeping the user interface Activitys very thin in their logic. Below is a diagram which illustrates communication from the user interface at the top, to the core Blue Nomad implementation pieces at the bottom.

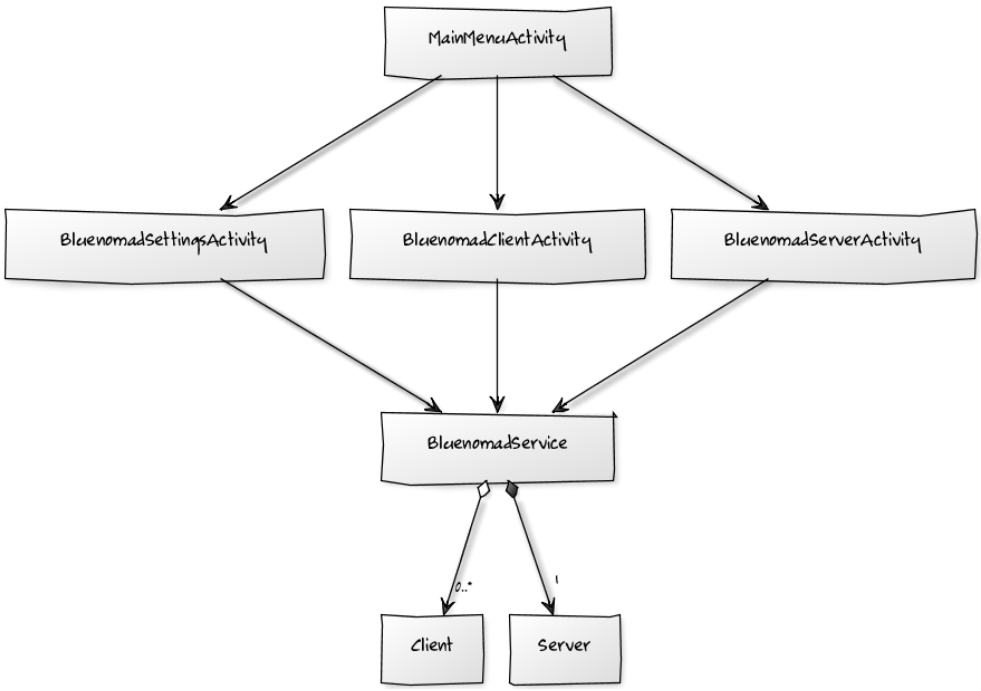

Figure 4-6: Blue Nomad Android Component Diagram

CHAPTER 5
RESULTS

Screenshots

Below are a series of screenshots from the Blue Nomad Android app.

Main Menu

The following image is a screenshot of the Blue Nomad main menu.

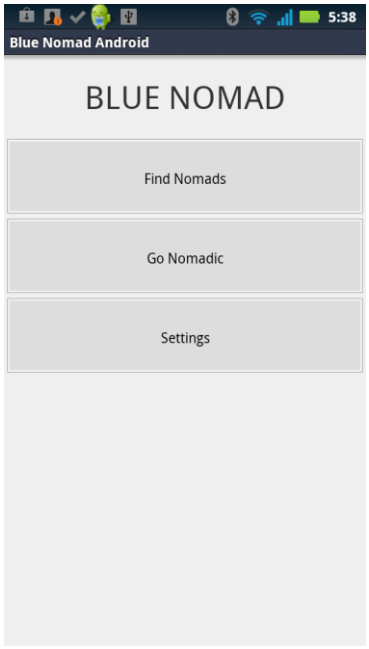

Figure 5-1: Screenshot of Blue Nomad Android Main Menu

App As A Server

The following sequence of images are screenshots illustrating the process of using the Blue Nomad app as a server (offering files to other devices). The user enters the "Go Nomadic" page, sets the device to be "discoverable," and then waits for another device to connect to this server. The device will only be discoverable for 120 seconds as per an Android limitation.

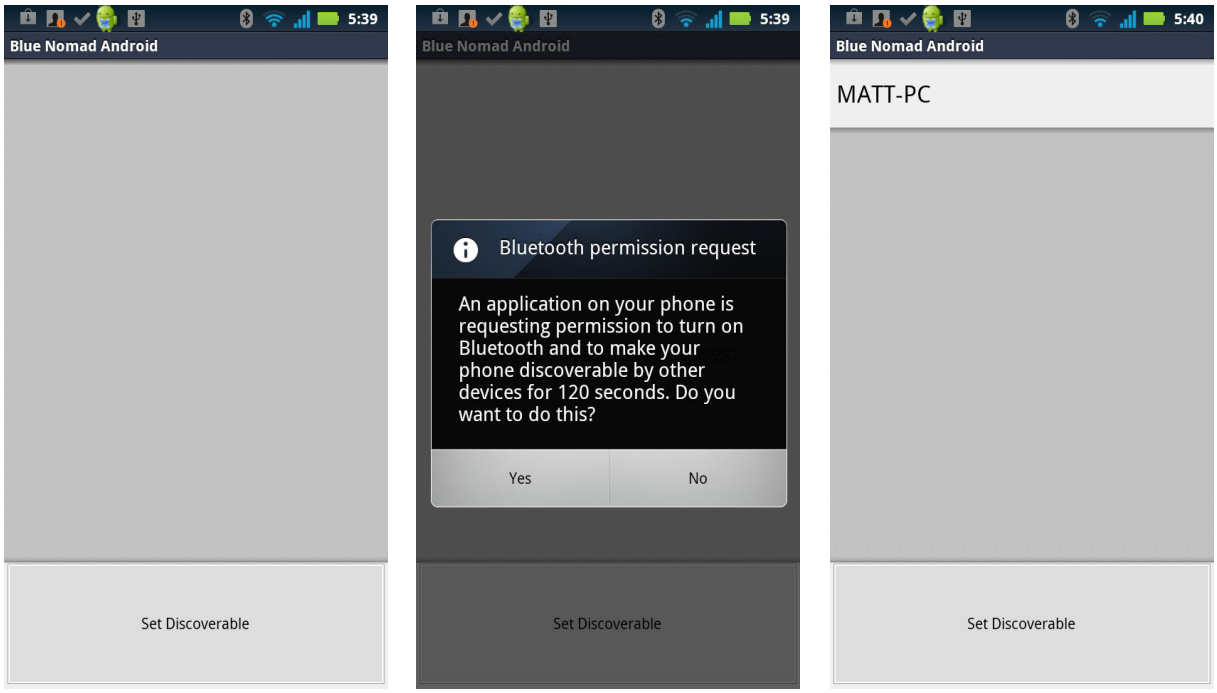

Figure 5-2: Screenshots of Blue Nomad Android as Server

App As A Client

The following sequence of images are screenshots illustrating the process of using the Blue Nomad app as a client (downloading files from other devices). First, the user enters the "Find Nomads" page. Then the user selects "Find Nomads" to begin discovering nearby devices offering the Blue Nomad service. For every device located, the Blue Nomad app attempts to connect to that device and begin Blue Nomad communication. If this connection/communication is successful, then an entry is placed in a list for that server; the list entry displays the name of the server next to the server's icon. Finally, the user selects a server that he/she wishes to download from. Once selected, a new view pops up which shows a list of every file available on the selected server. Selecting a file in the list will automatically download that file to the user's device.

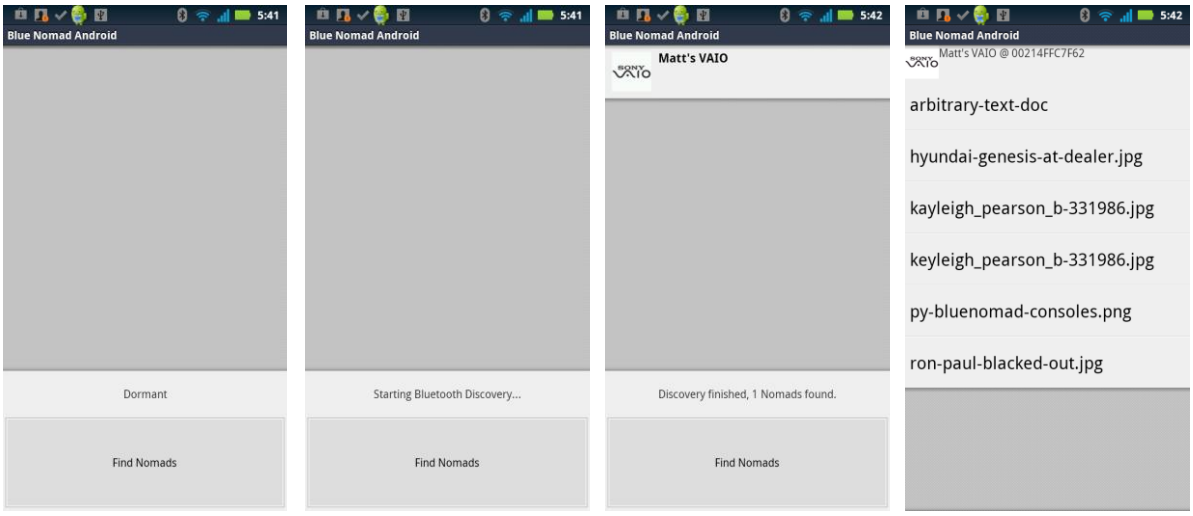

Figure 5-3: Screenshots of Blue Nomad Android as Client

Bluetooth Limitations

The Bluetooth protocol is designed to use very little power. Therefore, Bluetooth connections are highly limited both in range and bandwidth. Below are some Bluetooth metrics:

Power Level	Broadcast Range
Lowest	~1m
Typical	~10m
Highest	~100m

Table 5-1: Common Bluetooth Broadcast Ranges

Bluetooth Specification	Typical Transfer Rate
1.2	up to 721 kbit/s
2.X + EDR	~3 Mbit/s
3.0 + HS	up to 24 Mbit/s

Table 5-2: Bluetooth Transfer Rates by Protocol Version

Blue Nomad testing showed the following typical transfer rates:

Server	Client	Average Transfer Rate
Droid X2 Phone	Windows 7 Laptop (Bluetooth built-in)	~300 kbit/s
Droid X2 Phone	Ubuntu 10 Desktop (Bluetooth dongle, 2.1 + EDR)	~900 kbit/s

Ubuntu 10 Laptop (Bluetooth dongle, 2.1 + EDR)	Ubuntu 10 Desktop (Bluetooth dongle, 2.1 + EDR)	~500 kbit/s

Table 5-3: Average Blue Nomad Transfer Rates by Device

The tested transfer rates were far below the published speeds. Even transferring data between both Ubuntu machines, which were both using new Bluetooth 2.1 + EDR dongles pushed a lower average transfer rate than the published max for Bluetooth 1.2. One surprise was that transferring files from the Droid X2 to the Ubuntu Desktop actually out-performed the Ubuntu/Ubuntu transfer. Observations of eratic speed changes during testing lead me to believe that combinations of proximity, physical obstacles, and computer speed may all play a factor in the actual transfer rate. Regardless of the reason, the Bluetooth transfer rate is a major detractor to Nomadic Computing in that it prevents any sizable file from being transferred in a trivial period of time. For example, transferring a picture taken with the Droid X2 to the Windows 7 Laptop took in excess of 40 seconds; this is far too long for common social interaction.

Testing also revealed an unwieldy service discovery system. Local devices were almost always discovered successfully, but when searching for the Blue Nomad service it often took multiple executions of the discovery process to actually recognize the service. It is unclear exactly why this is the case, but Bluetooth is a non-deterministic protocol in that results during discovery will vary by circumstance.

Android Limitations

The Android platform introduced an unfortunate limitation for Bluetooth discovery. Not only does Android force the user to accept a dialog message to enable Bluetooth, but the Android system will not allow the device to remain discoverable for more than 120 seconds. This is problematic because realizing Nomadic Computing requires

perpetual discovery of devices - the goal is to connect people before they meet, not after.

The nature of Bluetooth as a low power system, along with some security vulnerabilities makes it unlikely that this 120 second discovery limit will be lifted. Therefore, Nomadic Computing may require a more continuous connection such as a Wifi connection. A Wifi connection would not necessitate the World Wide Web, but it does typically require additional hardware such as a router. However, the Bluetooth 3.0 + HS specification is interesting in that it proposes using Bluetooth to negotiate a connection and then use a high speed 802.11 connection for data transfer. This system may still impose the discovery limit, but directing Bluetooth along this route is promising in terms of obtaining a Wifi system that can easily create local P2P connections.

CHAPTER 6
CONCLUSION

Legal & Moral Implications

File sharing brings with it certain legal and moral implications. Using file sharing simply to transport a file you own, or even to broadcast a file to which you own the rights, is not an issue, either morally or legally. However, the situation changes drastically when a user shares a file to which he/she does not have a copyright or a distribution license.

Legally speaking, it is unlikely that Blue Nomad or any other Nomadic Computing platform would be held liable for any copyright infringement on the part of its users. Considering the case A&M Records v. Napster (2001), the courts determined that finding Napster guilty of copyright infringement required "personal conduct that encourages or assists the infringement." (*A&M Records v. Napster, Inc.*, 2001). Such "encouragement" would not occur with a generic Nomadic Computing platform.

Users of a Nomadic Computing platform, on the other hand, could be held liable for the sharing/broadcasting of copyrighted materials. In the U.S., copyright grants *exclusive* rights to the holder to reproduce, distribute, and transmit the copyrighted material (Copyright Act (1976)). Hence, taking any of the aforementioned actions with popular music, movies, books, etc. would violate copyright law.

Moralistically, users must deal with a situation similar to the legal situation. Clearly, moving, sharing, and broadcasting files to which a user owns the copyright, or to which the user is permitted to take such actions is completely acceptable. Likewise, blatant copyright violation with goods from honest producers is a moral violation within Western culture. A grey moral area tends to arise only when copyright infringement is performed

against a producer who is perceived to be dishonest. For example, some producers may engage in anti-competitive means such as price controls, running sweat shops, pursuing monopolistic control, etc. In these cases many users may consider copyright infringement to be a form of civil disobedience while the law continues to see it as a direct violation of intellectual property rights. While Nomadic Computing might change the landscape of copyright infringement, it is unlikely that Nomadic Computing presents a system which cannot be treated with existing case law.

The Future of Nomadic Computing

The underlying technical feat of connecting devices within a locality and transferring information between them has been shown to be possible. Disregarding an unwieldy discovery process, and a bothersome discovery time limit for Bluetooth, the process is not just possible, but reasonable for a production system. The current project could probably be adapted into a helpful utility app for the Android platform (and extended to iOS and Windows 8 if desired). However, to realize Nomadic Computing, there is still much work to be done.

Nomadic Computing will necessitate never-ending discovery of other devices. This discovery process may require use of another technology like Wifi, or it may require a new technology stack altogether - perhaps a combination of Bluetooth and Wifi. Additionally, the protocol will need to grow considerably, especially when considering the possible extension points of Nomadic Computing.

Nomadic Computing could provide a new implementation for a system like the Internet. Consider a content search algorithm that does not search web servers all over the world, but instead searches the contents of every device that you are able to reach. Initially this seems unnecessary because you would not expect to be surrounded by

more than a dozen or so devices at any one time. The situation becomes interesting when you consider that your radius of connection is expanded by the connection radii of everyone around you. Your bubble of connectivity overlaps a dozen other bubbles, but then those dozen bubbles may themselves overlap another dozen. Depending on the density of users and the range of communication, you could piggy-back signals around the block, or across a city. Suddenly there is a very complex infrastructure which arises from simply connecting a device to all of those around it. The network almost pervades society. Many experts agree that the future of computing is mobile, in fact, many expect to see computing become pervasive. Nomadic Computing tells us how we will get there.

APPENDIX A
BLUETOOTH AND BLUE NOMAD REFERENCE

Bluetooth Libraries By Platform

Bluetooth is a protocol and therefore is implemented by different libraries on different platforms. For reference purposes, a few major platforms and associate major Bluetooth libraries are provided below:

Platform	*Bluetooth Library*
Java SE	BlueCove
Python (on BlueZ stack)	PyBlueZ

Blue Nomad Protocol Specification

The following figures define various aspects of the Blue Nomad protocol including return codes, commands, and Bluetooth identification information.

Return Codes By Topic

Code Group	*Group Meaning*
1XX	Positive Preliminary reply
2XX	Positive Completion reply
3XX	Positive Intermediate reply
4XX	Transient Negative Completion reply
5XX	Permanent Negative Completion reply
X0X	Syntax & Misc.
X1X	Server Information
X2X	Filesystem Information

Server Return Codes

Codes	*Explanation*
100	Response indicating general acceptance
120	Response indicating start of file list
121	Response indicating request is good, binary transfer about to begin
200	Response indicating general completion
210	Response providing info about the server
221	Response providing info about a file
420	Response indicating desired file is busy
510	Response indicating desired server info is unavailable

520	Response indicating desired file does not exist
500	Response indicating command breaks protocol syntax
501	Response indicating desired filename has bad syntax

Client > Server Commands

Protocol Command	*Explanation*
GET_ID	GET_ID
GET_NAME	GET_NAME
GET_ICON	GET_ICON
GET_FILELIST	GET_FILELIST
GET_FILE	"ET_FILE
CONN_CLOSE	CLOSE

Blue Nomad Bluetooth Properties

Property	*Value*
Service Name	Blue Nomad
Service UUID	1e0ca4ea-299d-4335-93eb-27fcfe7fa848

LIST OF REFERENCES

A history of bittorrent. (2011, May 12). Retrieved from http://visual.ly/history-bittorrent

A&M Records v. Napster, Inc., 239 F.3d 1004, 1019 (2001).

Copyright Act of 1976, 17 U.S.C. §106 (2011).

Developer guide. (n.d.). Retrieved from https://developers.google.com/protocol-buffers/docs/overview

The e4x approach to xml processing. In Programming ActionScript 3.0 for Flash Adobe. Retrieved from http://help.adobe.com/en_US/ActionScript/3.0_ProgrammingAS3/WS5b3ccc516d4fbf351e63e3d118a9b90204-7e72.html

Even worse than sopa: New cispa cybersecurity bill will censor the web. (2012, April 04). Retrieved from http://rt.com/usa/news/cispa-bill-sopa-internet-175/

Jung, S. (2007). Bluetorrent: Cooperative content sharing for bluetooth users. Pervasive and Mobile Computing, 3(6), 609-634. doi: 10.1016/j.pmcj.2007.06.003

Koneri, A. (2004). Ftp client explained (in java). Retrieved from http://www.angelfire.com/or/abhilash/site/articles/ftp/ftp.html

Programming guide. (n.d.). Retrieved from http://xerces.apache.org/xerces-c/program-3.html

Protect ip / sopa breaks the internet. (n.d.). Retrieved from http://fightforthefuture.org/pipa

Protocol buffers. (n.d.). Retrieved from https://developers.google.com/protocol-buffers/

Rowan, N. (2006). A detailed history of the silk roads. Retrieved from http://www.travelthesilkroad.org/content/view/15/29/

Sopa and pipa anti-piracy bills controversy explained. (2012, January 17). Retrieved from http://www.bbc.co.uk/news/technology-16596577

Tyson, J. (2000, October 30). How the old napster worked. Retrieved from http://computer.howstuffworks.com/napster1.htm

BIOGRAPHICAL SKETCH

Matthew Carroll is a software developer, entrepreneur, and libertarian. He received both a B.S. in Computer Science and a B.A. in Mathematics from the University of Florida in 2010. Matthew has worked professionally as a Flash developer, Android developer, project manager, and CEO. Matthew's passion is the creative process; his vision is a world of innovation.